

FRAUDWEILER

Implementing a plagiarism detection system



Technical Writing Group 15C

Ana Băltărețu
Andra-Georgiana Sav
Renāts Jurševskis
Radu Gaghi
Octav Pocola

TA: Sára Juhošová

Coach: Sebastian Proksch

Client: Educational Innovation Projects

June 2021

i Preface

This application was developed as part of the CSE2000 Software Project course at the Delft University of Technology. The Software Project is carried out by second-year bachelor students of Computer Science in groups of five and aims to provide the students a real-life experience of developing an application. During the project, each team must develop a product that was requested by a company and solves a real-world problem. Furthermore, another objective of the project is to have the students document their process.

The project that our group chose has the goal of replacing the old plagiarism detection system currently available at the Delft University of Technology. The new system aims to drastically reduce the time required for running a plagiarism detection check while providing flexibility when adding more algorithms.

Our team consists almost entirely of teaching assistants and mentors, so we have all been involved in the learning process at TU Delft. This is one of the main reasons we chose this project: we thought we could improve the quality of the learning process at TU Delft by designing and implementing this system.

This report documents the development process of the application.

Lastly, we would like to thank all the people that helped us during this project for all of the support they provided. This includes our client, Taico Aerts, our TA, Sára Juhošová, our coach, Sebastian Proksch, and our technical writing lecturer, Wim Blokzijl. Some additional people who we would like to thank are O.W. Visser and Kevin Chong for all the feedback they provided.

Delft, 17 June 2021

Ana Băltărețu, Andra-Georgiana Sav, Renāts Jurševskis, Radu Gaghi, Octav Pocola

ii Summary

Due to the arrival of the COVID-19 pandemic and the sudden move to an online education environment, the chances of students plagiarising have highly increased. The lack of a centralized Plagiarism Detection System at the Delft University of Technology, as well as the laborious fraud-checking work the teachers had to do, determined the need for developing an automated system. The purpose of this report is to present the development process of Fraudweiler, a plagiarism detection system built during the Software Project.

At the client's request, Educational Innovation Projects, it was established that this new system will be extensible and flexible with regard to different kinds of imported assignments, as well as to adding various types of algorithms. Ultimately, the goal of the system was to completely replace the previous one created at TU Delft, ControlV, which was not entirely fulfilling the needs of the university. Therefore, it was decided it was best to start it from scratch and design it properly, rather than having to modify the whole system itself.

A feasibility study and a study regarding the risk analysis concluded that building such a system is indeed viable and would be highly beneficial in an academic environment.

The requirements of the application were made to align with the client's expectations, considering at the same time the different stakeholders involved. In that sense, a survey was conducted to gather data about what the final system should ideally look like.

Although all of the requirements discussed during the elicitation process were indeed valuable and useful, given the time constraint they had to be prioritized. For that, the MoSCoW [1] method was used, to ensure the most essential features (i.e., uploading assignment folders, automatically generating and exporting reports, LabraCORE integration, comparing submissions side-by-side, setting default algorithms) were implemented. Therefore, according to the planning, the must-haves and should-haves were fully done. Due to the short amount of time available, it was established that some of the requirements were out of the scope of this project (e.g., implementing more complex algorithms and visualizing their results).

As the client requested, the system was created with flexibility in mind. Thus, the design choices made, such as the usage of the decorator design pattern, allow easily adding more algorithms and folder structures. Moreover, the design showcases the user interface in accordance with other TU Delft's systems, intended to be easy to follow and understand. Since the system handles private and sensitive data, ethical implications were discussed, such as storing submissions that are the intellectual property of students and access to their data. With that in mind, GDPR [2] policies were closely studied and incorporated within the application.

The implementation of this system required a thorough organization. To this extent, Scrum methodology was used, and an early-testing approach was adopted. Furthermore, communication with all the involved parties, such as the client and the TA, was established almost weekly, to ensure smooth cooperation. Building the functionality required using several frameworks, such as Spring for the back-end, and Thymeleaf for the front-end, which is currently used by other TU Delft's systems. Moreover, the relevant data about students, courses, and teachers is retrieved from the LabraCORE system.

Although the implementation of advanced algorithms was out of the scope of this project, the Diff algorithm (text-based) and the JPlag algorithm(token-based) [3] were still added, which makes the application already functional.

The overall performance was hard to evaluate, so it was tested mainly from the algorithm run-times and database retrieval perspectives.

The system offers a great range of useful features, such as a folder-like structure for assignments and results, manual and automatic highlighting of submissions, as well as the possibility to remove template code and to include revisions in the fraud checks. However, these come at the cost of having a larger run-time when running checks. With that in mind, user evaluations were created such that the product could be improved before delivering its final version.

Lastly, although a piece of software is never complete, the system can be already used to detect fraudulent submissions in examinations. To highlight the most important features, here is a list of some of them:

1. Uploading folders with assignments.
2. Making a new fraud check and selecting submissions, algorithms, and different settings.
3. Viewing the results of a check in an organized manner.
4. Viewing groups of submissions and having the ability to highlight interest points.
5. Creating a fraud report that contains user information, and comments from the teacher.
6. Exporting reports which contain the previously selected details and evidence.

Therefore, the system adds value to the quality of the education provided at TU Delft, as it automates the laborious work teachers had to previously do. Additionally, because it is very flexible, future improvements can easily be integrated with the system. In that sense, several recommendations for further development were made, such as adding more algorithms to improve the overall accuracy of a fraud check, improving the user-friendliness, and constantly complying with the GDPR policies.

Contents

i	Preface	2
ii	Summary	3
1	Introduction	7
2	Problem analysis	8
2.1	Impact on education	8
2.2	Plagiarism detection systems	8
2.2.1	Plagiarism detection in documents	8
2.2.2	Plagiarism detection in code	8
2.3	Already existing products	9
2.3.1	Turnitin	9
2.3.2	WCopyFind	9
2.3.3	Plague	9
2.3.4	YAP	9
2.4	Stakeholders	10
2.5	Feasibility study	10
2.5.1	Technical feasibility	11
2.5.2	Operational feasibility	11
2.5.3	Schedule feasibility	11
2.5.4	Legal feasibility	11
2.5.5	Conclusion on feasibility	12
2.6	Risk analysis	12
2.6.1	Dependency on other systems	12
2.6.2	Data collection	12
2.6.3	System performance problems	12
2.6.4	Unexpected changes in requirements	13
3	Requirements of the plagiarism detection system	14
3.1	Requirement prioritization	14
3.2	Explanation of terms	14
3.3	Non-functional requirements	15
3.4	Functional requirements	15
3.4.1	Must have	15
3.4.2	Should have	16
3.4.3	Could have	17
3.4.4	Won't have	17
3.5	Requirements elicitation	17
3.5.1	Client interview	18
3.5.2	Survey	18
4	Designing the plagiarism detection system	19
4.1	Flexibility of the system	19
4.2	Architecture	19
4.2.1	Algorithm design	19
4.2.2	Database design	20
4.2.3	Design patterns	21
4.2.4	User interface design	22
4.2.5	Security	23
4.3	Ethical implications	23
4.3.1	Protection of private data	23
4.3.2	Social stigma as a result of identifiable data	23
4.3.3	Intellectual property	24

4.3.4	Measures to ensure GDPR compliance and data protection	24
5	Implementation of the plagiarism detection system	25
5.1	Development methodology	25
5.1.1	Processes	25
5.1.2	Communication	25
5.1.3	GitLab	26
5.2	Building the system	27
5.2.1	Spring framework	27
5.2.2	Graphical User Interface	28
5.2.3	LabraCORE	29
5.2.4	Algorithms	30
5.3	Performance	30
5.3.1	Plagiarism detection algorithms	30
5.3.2	Data retrieval	31
6	Product discussion and future enhancements	32
6.1	Strengths and weaknesses of the system	32
6.2	Possible future improvements	36
6.3	Product evaluation	37
6.3.1	User interface evaluation	37
6.3.2	Performance evaluation	37
7	Conclusions and recommendations	39
	References	40
A	Level of code Plagiarism	42
B	Surveys	42
B.1	Requirements survey	42
B.2	User interface (UI) Survey 1	43
B.3	User interface (UI) Survey 2	45
C	Contribution	47
D	Templates	48
E	Graphical User Interface (Design)	49
F	Graphical User Interface (Implementation)	58

1 Introduction

In recent years there has been a steep increase in the number of online assignments and exams and, in the past year, this increase has been boosted even more by the appearance of the COVID-19 pandemic and the move to an entirely online education environment. This change is specifically significant for universities since it is comparatively easier for students to plagiarize in an online setting. Fraudulent activities have a far-reaching impact both on the students and the teachers, constituting a highly demotivating factor for students who do not commit fraud, and it can even damage the reputation of universities. Preventing plagiarism is an important step that needs to be taken, and at the moment, most of the work falls on the teachers responsible for a course, who have to spend countless hours manually checking submissions. The Delft University of Technology (TU Delft) is one of the countless universities affected by this issue, and, at the moment, there exists no centralized Plagiarism Detection System (PDS) that could be easily integrated with other platforms used by the TU Delft. Even though there exist some tools that can identify fraudulent work, none of them can be extended to incorporate the different features that the client, Educational Innovation Projects (EIP), has requested. The software will benefit both teachers and students, because it will ensure fair grading, by offering the guarantee that a student's work could be properly assessed, and it will decrease the time that a teacher spends checking submissions.

The goal of this report is to showcase the design choices, the development process, and the final result of building the plagiarism detection software. This report will explain in detail how the application was designed, it will describe the difficulties that have been encountered and the design choices that were made to solve these difficulties. Lastly, possibilities for future development will be presented. One of the features that was of high importance to the client was flexibility. It refers to the flexibility of importing various types of folders with different folder structures, as well as having a simple way of extending the existing collection of algorithms, and, to solve these requirements, a couple of design patterns were used. Another important requirement was creating a centralized platform that is integrated with LabraCORE and that can automatically generate fraud reports after running checks on large amounts of submissions. During the building process the most common ways of plagiarizing, such as copying code from other students, as well as from the Internet, were taken into account. The different levels of plagiarism were analyzed to have a better overview of what factors require more attention. At the beginning of the project, the executive decision of building the system from scratch was taken, instead of extending the existing platform. This choice was made in order to focus on the flexibility requirement, but, because of the short duration of the project, this meant that there wasn't enough time to implement many algorithms.

This report is structured as follows: Chapter 2 analyses how the product will influence education and explores the effects of the software on direct and indirect stakeholders; Chapter 3 includes the requirements of the software, which are divided into functional and non-functional requirements; Chapter 4 presents the design details that have been considered and it explains how the developers have designed the product by focusing on its flexibility; Chapter 5 explains the approach and the process throughout the project, and it includes technical details such as how the performance requirement was taken into account; Chapter 6 analyses the product's strengths and weaknesses and showcases some possible improvements for the product; Chapter 7 presents the conclusion and the recommendations we give to future users and developers working with our software.

2 Problem analysis

This chapter will give context to the problem of automating the plagiarism detection process. This will be achieved by firstly looking at the impact that plagiarism has on higher education (2.1). Then, there is an overview of how plagiarism detection is accomplished (2.2), both in documents and in code. An analysis of already existing products (2.3) is presented, as it is vital to understand their strengths and weaknesses in order to build a viable system. An analysis of different stakeholders and their interests is also included in this chapter (2.4). And, to give the reader a more complete picture, a feasibility study (2.5) and a risk analysis (2.6) are also provided.

2.1 Impact on education

Plagiarism in education can be a highly demotivating factor for both students and teachers if not addressed sufficiently, as students who plagiarized could gain an unfair advantage, generating a negative impact on the learning process [4]. In her paper on Plagiarism in Education, Ryan states that she caught 18% of her students plagiarizing, during a five year period [5]. Another study, which surveyed students from the Slovak University of Technology in Bratislava, found that 33% of students admitted that they plagiarized while 63% of students admitted to giving their work to others [6]. These are alarming statistics, that have probably become even worse due to the migration of most of the exams to an online medium because of the COVID-19 pandemic. The amount of work necessary for detecting all these instances of plagiarism can leave the teaching staff unmotivated. However, in the same survey that was mentioned before [6], 72% of the teaching staff reported that an automatic plagiarism detection tool would greatly increase their motivation to properly check for fraud.

2.2 Plagiarism detection systems

A plagiarism detection system (PDS) is a program or a set of programs that automates the task of plagiarism detection by finding similarities between students' submissions. One can identify two distinct types of PDSs, one for text-based documents and one for code-based documents. The upcoming subsections will describe the particularities of each of these types of PDSs.

2.2.1 Plagiarism detection in documents

Detecting plagiarism in text consists of finding similarities between two text-based documents. While word-for-word plagiarism might be easy to detect, this becomes difficult when sentences are rephrased. Furthermore, some techniques exist to mislead PDSs, such as: replacing English letters with Cyrillic letters which look the same (e.g. replace e with the Russian letter e), inserting white characters, turning in assignments as images, etc.

One can classify the PDSs for documents in two distinct categories: web-enabled and standalone. Web-enabled PDSs are usually hosted online (e.g. Turnitin) [4]. A standalone PDS usually needs to be installed on a specific computer (e.g. WCopyFind).

2.2.2 Plagiarism detection in code

Detecting plagiarism in code consists of finding similarities between code-based submissions. A variety of approaches have been proposed already [7]. The most common technique for detecting code duplication consists of creating a fingerprint for each submission, often using some form of tokenization ¹. These fingerprints ² are then matched using classical string matching algorithms

¹Tokenization is the process of creating an identifier (token) for a submission. This token can then be used to identify the original submission. Note that tokens don't have to be unique.

²A fingerprint is a unique token.

[4] [7]. As in the case of document-based PDSs, they can also be classified as web-enabled or standalone. Different levels of code plagiarism have been identified by Faidhi and Robinson in their study [8] (Appendix A).

2.3 Already existing products

There is a large number of already existing solutions, as one could imagine, given that the field of plagiarism detection is an already mature field, with a lot of comparisons available [4] [9] [10]. In the following subsections, several qualities and drawbacks of the most well-known Plagiarism Detection Systems will be analyzed in order to better understand the problem the development team is facing.

2.3.1 Turnitin

Turnitin is the most popular commercially available PDS. It is a subscription-based, web-enabled system (i.e. it is hosted online). It compares a student submission with multiple online databases: a proprietary database, academic databases, student databases (i.e. databases containing past students' assignments). Additionally, web crawlers are used to continually expand their databases. It is compatible with educational platforms such as Google BlackBoard [11], Moodle [12], and also Brightspace [13].

One of the biggest downsides of Turnitin is the fact that students' submissions are saved in an external database. This creates a lot of legal problems, as the United States Government could in theory access this data under The Patriot Act [14]. Furthermore, multiple vulnerabilities have been documented (e.g. character encoding, rearranged glyphs ³, text replacement by Bezier curves ⁴, etc.) [15].

2.3.2 WCopyFind

WCopyFind [16] is an open-source program for Windows systems developed by Dr. Lou Bloomfield at the University of Virginia. It checks a batch of submissions for content similarity and is highly configurable through a set of parameters.

The downside of this is that the parameters require a high level of understanding of the system. Furthermore, the initial parameters highly influence the results, so the system requires extensive knowledge in order to be properly used.

2.3.3 Plague

Plague[17] is a web-enabled system, that checks one submission against an array of online databases, generating a plagiarism report. It is a free service, however, it only allows for one document to be uploaded at a time. Also, the time it takes to run a comparison on one submission is undeniably high (over one minute per submission pairs). This makes this PDS very time-consuming to use with a large batch of students. The initial Plague algorithm could only evaluate source code written in C. There exists a newer version, JPlag [3], that can be used to detect similarities in source code written in C, C++, Java, and Scheme.

2.3.4 YAP

YAP was developed based on the previously described PDS, Plague, and works in a similar way [18]. YAP hugely improves the running time of Plague. The algorithm was further refined into

³There are symbols that look the same but are represented by different numbers in Unicode (e.g. the English letters B and the Cyrillic letter B which look the same but are different letters).

⁴A Bezier curve is a parametric curve used in computer graphics. It can be used to replace a letter with a drawing of the same letter. This will appear identical to the naked eye, but not to a computer program.

YAP2 and YAP3 [19]. The latest version can be used to detect similarities in both code and text documents.

All the YAP family of algorithms have two essential parts:

1. generating a token for each source code
2. comparing the tokens between them

The increase in speed comes from this tokenization of files, as the document is not compared in its entirety.

2.4 Stakeholders

When designing a solution for a problem, it is important to know who you are solving it for and understanding the context. By identifying the different stakeholders one can see the problem from more points of view and create a better product.

Direct stakeholders. The direct stakeholders which were identified in this situation are the teaching staff, the Board of Examiners, and the students. Each of these stakeholders has different expectations and concerns about the system:

1. The **teaching staff's** main interest would be to automate the plagiarism detection task as much as possible and increase the effectiveness of fraud checking to deter students from fraudulent practices.
2. The **Board of Examiners** would only interact with the system through the generated fraud reports. These reports will have to contain all the relevant information that they might need, such as the names of the students and the parts of the assignment that are suspected of being plagiarized. The report will be generated after a given template and will follow a specific format.
3. The only direct interaction the **students** have with the system is when they get flagged for fraud and are notified by the Board of Examiners. However, their work is, after all, their intellectual property, so it should be handled with care. Students are subject to a large amount of stress and being reported for fraud only adds to it. The ethical aspect of this is discussed in chapter 4 . On the other hand, the students who do commit fraud have a negative stake in the project (i.e. they are negatively impacted by the success of the project). This type of student might want to mislead the system through various techniques (some of which were stated in 2.2.1).

Indirect stakeholders. The indirect stakeholders which were identified are the developers of other services used by TU Delft (e.g. Labrador or WebLab), as the system will need to be integrated with the rest of the university's technological infrastructure. The system should allow for further expansion with more algorithms and import from other sources with ease.

2.5 Feasibility study

After establishing the functionality of the system together with the client, the developers have analyzed several aspects, such as technical, operational, legal, and organizational ones. The purpose of this study was to see if the proposed solution was indeed viable.

2.5.1 Technical feasibility

The project is a web application, and the following technologies will be used in the process of designing and creating it: Java, Spring, MySQL, Thymeleaf, Lucidchart (for database schema), and Figma (for mock-ups). All of these technologies are available online and are free to use. Moreover, each developer has the necessary prior experience with most of these technologies. Considering the resources needed, these are:

- Programming devices (laptops)
- Technological tools (above-mentioned)
- Software developers (five available)

Thus, the resources needed are available for this project, which makes it technologically feasible.

2.5.2 Operational feasibility

There currently exists a PDS that aims to provide the necessary fairness of examinations. However, due to the arrival of the COVID-19 pandemic and the sudden change to online education, the rate of fraud has greatly increased. Thus, both teachers and students need to be assured, using a new and improved system, that everyone is correctly and fairly examined.

Firstly, the system aims to provide more flexibility, in the sense that it will be easily extensible and it will allow for adding new algorithms at any point in time. This is especially useful because there is ongoing research on this theme, thus, if there is a better approach found, it can be easily integrated with the fraud system.

Secondly, the system will be highly customizable, meaning that checks can be done by the user using different types of pre-selected algorithms. In this way, the output will be more indicative, because it will take into consideration different aspects of the submissions, which might not be noticed during a manual check.

Considering these, it is believed that after deployment, the system will be easily maintainable and usable, making it operationally feasible.

2.5.3 Schedule feasibility

After conducting the requirements elicitation with the client, the major features of the application that is to be developed were established. These were split into four categories, namely "must haves", "should haves", "could haves" and "won't haves". Since the time limit to do the project is ten weeks, the "must haves" requirements will be prioritized, which are essential to the client, followed by the "should haves", and then, based on the time left, some of the "could haves" might be implemented as well. Because the team is composed of five members, and each is expected to work approximately 36 hours per week, it is believed that it is feasible to finish at least the "must haves" and the "should haves" within the given time.

2.5.4 Legal feasibility

In terms of legal aspects, the application complies with the current GDPR (General Data Protection Regulation) policies [2]. The developers have studied the issue closely, together with the client and the other involved parties (such as the LabraCORE team - which provides the means of authentication, and Submit – which is one of the providers of students' data and submissions). Thus, the study has shown that the project is legally feasible.

2.5.5 Conclusion on feasibility

Given the fact that the project is technologically, legally, and operationally feasible, it is believed it can be safely implemented given the time constraint of 10 weeks.

2.6 Risk analysis

In software development, various risks can arise at any point in time. Therefore, an effort has been made to identify and evaluate them, as well as plan ahead some possible solutions and preventive measures. In the upcoming sections, several risks will be discussed and analyzed in more detail.

2.6.1 Dependency on other systems

The product will need to connect with multiple other platforms (i.e., WebLab, LabraCORE, Submit), so the development process could encounter issues regarding the interaction of the various systems. If one of these systems were to be modified, it could cause major setbacks in the development process. Therefore, this creates dependencies with factors that cannot be controlled like the decisions of other development teams. Another important factor that needs to be considered is that the development of the product could require changes to be made in some of the systems that it interacts with. This issue could be even more severe, as the successful and timely completion of the product would depend on other development teams being open to the requested modifications and completing them in a short amount of time. To minimize the likelihood of encountering these problems, meetings will be held with the developers of these systems and discussing any possible setbacks. In addition, the development team will be staying up to date with the release of new updates and changes to the platforms.

2.6.2 Data collection

Modern data collection laws are very strict about the storage of personal user data. With the introduction of the GDPR (General Data Protection Regulation) [2], new sanctions have been established for violating the regulations regarding data misuse or data loss. As the product will be storing sensitive information like student submissions and information about potential fraud cases, it must be certain that the regulations are followed and that data is stored correctly and securely. While both the content of the GDPR and the implications of violating it have already been researched and discussed, it could still cause issues and delays during the development process. In order to minimize the risk, the development team has chosen to research the legal implications of GDPR even further and to take additional precautions by having discussions with more experienced developers from the WebLab or LabraCORE development teams.

2.6.3 System performance problems

The product will need to store a large amount of data. Therefore, during the development process, a situation could be encountered where the data retrieval from the database is not performing well enough. In addition, the plagiarism detection algorithms could also cause performance problems if not implemented well or not appropriate for the necessary scalability. In order to mitigate these issues, it might be necessary to make modifications that could cause large delays in the development of the project. To minimize the risk of encountering performance-related problems proactive steps will be taken by designing the product with performance in mind rather than reacting to performance issues only when they are encountered. This will be accomplished by thoroughly researching the time and space complexity of each of the algorithms, estimating the expected number of records in the database, and closely monitoring the performance of the product during the development process.

2.6.4 Unexpected changes in requirements

Although a comprehensive list of requirements has already been compiled, some changes could still be made later in the project either due to technical limitations or modifications from the client. The change of requirements can have various effects on the development process depending on the severity of the necessary adjustments. Although most changes should not delay the development too much, there is also a possibility that adapting to the modified requirements involves a complete overhaul of the system architecture. Therefore, many precautions are being taken to minimize the chance of unexpected changes in requirements like maintaining proper communication with the client and analyzing the feasibility of the product. In addition, agile development practices are being followed that allow for the flexibility of adapting to changing requirements.

3 Requirements of the plagiarism detection system

In this chapter, a list of requirements that have been gathered after interviewing the client and surveying potential users of the platform will be presented. The chapter begins with a description of how the requirements were prioritized (3.1), and continues with an explanation of the terms (3.2) that are used in the requirements description. These sections are followed by the actual requirements, which were split into two categories: Non-Functional Requirements (3.3), and Functional Requirements (3.4). The Functional Requirements were further divided and prioritized using the MoSCoW [1] method. The chapter ends with a description of the requirements elicitation process (3.5), which describes how the client was interviewed and how the survey of potential users was conducted.

3.1 Requirement prioritization

Although all of the requirements gathered from the elicitation process are valuable, some have a higher priority than others. The importance of a requirement has to be evaluated based on many factors like the wishes of the client, the technical feasibility, and the trade-off between the cost and the amount of gained value. This priority can then be used to determine which requirements are integral to the product and which ones are out of scope and will not be implemented.

Therefore, it was important to determine the appropriate prioritization of requirements. To categorize the functional requirements based on their priority, the MoSCoW [1] method was chosen, with the "must haves" being the requirements with the highest priority, which must be finished by the end of the project to achieve a minimum viable product. These are then followed by the "should have" requirements which are not crucial for the delivery of the product but are still important. Similarly, the "could have" requirements are also not the main focus of the system, and would only be included if time and resources permit. Finally, "won't have" requirements are too far out of the scope of the project, so they will not be implemented.

3.2 Explanation of terms

Explanation of entities:

1. **Submission**, the response of a student to a question/problem/assignment.
2. **Assignment**, a question answered by multiple students that results in a folder containing students' submissions.
3. **Check**, a group of algorithms that use different metrics to compare a set of submissions, giving as output a list of results.
4. **Result**, a pair of two *submissions* that have a matching score, based on one algorithm.

Explanation of user types:

1. A **Student**, does not have access to the system, but it will be used within the system to display information such as name, or student number, which will be retrieved through LabraCORE.
2. A **Manager**, commonly referred to as a Head TA, has access to the system in course editions for which they have this role, but they do not have access to the fraud reports.
3. A **Teacher**, has access to all parts of a course edition they are part of.

3.3 Non-functional requirements

1. The system will have a clear setup guide, in the sense that a developer of the system could install and run the system by following it, without additional help.
2. The system will be GDPR-compliant, it will automatically delete submissions after a configurable amount of time.
3. The system will be secure in the sense that it will have authentication, authorization, and accountability.
4. The system will be scalable and it will be extendable, in the sense that it will allow future developers to easily add new algorithms, and it will contain a clear explanation on how to do this.
5. The system will be integrated with Weblab (i.e it will be able to import files from Weblab), LabraCORE (i.e it will use the API endpoints and the information stored in the database).
6. The system will have a clean and understandable User Interface, and to assure this, one or more experts will be asked to evaluate it.
7. The front-end should be easily modifiable and independent from the back-end.
8. The system will be built with Java on the back-end and Thymeleaf (JavaScript, HTML, and CSS) on the front-end.
9. The system must implement the login functionality through LabraDoor.

3.4 Functional requirements

The Functional Requirements were prioritized using the MoSCoW method [1], as explained in the Requirement prioritization section (3.1).

3.4.1 Must have

1. A Manager and a Teacher must be able to upload a batch of student submissions with the folder structure Course Edition/Assignment/Student submissions.
2. A Manager and a Teacher must be able to select the type of submission (e.g. text, code, etc.) when importing an assignment.
3. A Manager and a Teacher must be able to manually add other solutions, such as solutions available online and reference solutions to the pool of submissions.
4. When a Manager or a Teacher creates a new check, a default set of algorithms must be already selected.
5. A Manager and a Teacher must be able to select different algorithms for different submission types (e.g. text, code, etc.)
6. A Manager and a Teacher must be able to select multiple algorithms for an assignment when performing a check.
7. A Manager and a Teacher must be able to easily add new algorithms to a check, even after it is completed.
8. A Manager and a Teacher must be able to compare different course editions when performing a check if they have access to both course editions.

9. A Manager and a Teacher must be able to see relevant metrics from the results of applying the selected algorithms.
10. After a check is completed, a Manager and a Teacher must be able to see a list of matching submissions based on the results from the selected algorithms.
11. A Manager and a Teacher must be able to filter the submissions based on the name and/or a range of matching scores.
12. A Manager and a Teacher must be able to sort the submissions based on each metric, decreasing from the highest matching score.
13. A Manager and a Teacher must be able to sort the results based on the “smart score”, which is a combination of algorithm results.
14. A Manager and a Teacher must be able to create a report for the Board of Examiners following the official template for reporting fraud and including code snippets, which would result in saving the report in the database.
15. A Teacher must be able to access previously created fraud reports.

3.4.2 Should have

1. A Teacher should be able to export a batch of previously created fraud reports as PDFs.
2. A Teacher should be able to create a less specific report than the one for the Board of Examiners, for the student, notifying him/her of fraud, without including the evidence.
3. The system should support analyzing multiple revisions of the same submission.
4. The system should automatically identify major revisions and perform the fraud check on them.
5. A Manager and a Teacher should be able to upload input from different platforms (i.e. WebLab, Submit, etc.), to allow for easy importing of different input formats.
6. A Manager and a Teacher should be able to change the settings for which algorithms are selected by default for a *check*.
7. A Manager and a Teacher should be able to see previously calculated results for each triple of two submissions and one algorithm.
8. For a triple of two submissions and one algorithm, if the results have already been calculated and saved in the database, the system should reuse them, instead of re-calculating the results every time.
9. A Manager and a Teacher should be able to ignore a template when performing a check. (e.g., text, boilerplate code)
10. The system should automatically retrieve course/user information from LabraCORE.
11. A Manager and a Teacher should be able to see matching submissions side-by-side after running a fraud check.
12. A Manager and a Teacher should be able to add an algorithm to be run on a specific sub-assignment.
13. The system should support multiple metrics per algorithm to accept various algorithm output types.

3.4.3 Could have

1. A Manager and a Teacher could highlight code before creating the fraud report.
2. A Manager and a Teacher could see auto-highlighted code snippets when viewing the results of a check.
3. A Manager and a Teacher could see partial results while a check is running.
4. A Manager and a Teacher could create custom templates for which algorithms and parameters they want to have already selected for a check, which can be saved for later use.
5. A Manager and a Teacher could create custom templates for how the imported file structure would look like, to allow files to be stored in a different structure, which can be saved for later use.
6. A Manager and a Teacher could select a subset of submissions from the result and run another check on them (this will create a new check).
7. A Manager and a Teacher could be able to see matching submissions side-by-side with highlighting after running a fraud check.
8. A Manager and a Teacher could manually add additional submissions to a fraud report.
9. A Manager and a Teacher could customize the parameters used by algorithms when performing a check (e.g. changing the minimum length of what is considered a match).
10. The system could exclude empty submissions from the check, as to not get results with high similarity between them.
11. The system will automatically detect file types.
12. If multiple submissions match on the same criteria, a user can see them automatically grouped as a cluster.

3.4.4 Won't have

1. A Manager and a Teacher can mark submissions as “favorites” to be able to find them more easily when looking over a check again.
2. A Manager and a Teacher can write notes on the side when looking at a match.
3. A Manager and a Teacher can choose to look over the Abstract Syntax Tree (the code structure, but without variable names or comments) of a match to see for themselves where the structure matches.
4. For coding assignments, a user can choose to compare only on some parts of the code (i.e. a method), instead of comparing on the entire code.

3.5 Requirements elicitation

To discover the main drawbacks of existing solutions and explore the expectations of the project, it was necessary to discuss the platform not only with the client but also with other stakeholders of the product. Therefore, the elicitation process consisted of various stages such as interviewing the client and surveying the future users of the system.

3.5.1 Client interview

To understand the expectations of the project various meetings with the client were held where the requirements of the system were discussed. By making sure that the expectations of the client aligned with the extracted requirements increased the confidence of the development team in the understanding of the goals of the project while providing the client with a clear project plan. In addition, the final list of requirements was presented to the client for approval at the end of the requirements engineering process.

3.5.2 Survey

To make sure no important requirements were missed, a survey was conducted and received eight responses. The full list of questions has been included in Appendix B.1. In order to gather useful data, the survey was sent to a list of stakeholders who would be using our finished system like lecturers, head TAs, and our client. The stakeholders answered anonymously without the need to mention their role or involvement with the final product. However, information regarding the participants' experience with plagiarism detection systems was gathered.

The main issue that multiple stakeholders mentioned concerns the ability to add revisions to the check, and now it is even more clear that this issue should be focused on. From this survey, multiple suggestions have been received regarding improvements to the user experience, which have been added to their corresponding category in the list of requirements.

4 Designing the plagiarism detection system

This chapter aims to introduce the design of the application and the values the development team tried to uphold in the process of creating it. The first section will discuss the flexibility the system exhibits (4.1) when adding new algorithms or submissions. In the next section (4.2) fundamental design choices will be touched upon including the database and the user interface. Finally, the Ethical Implications (4.3) section will emphasize the importance of protecting private data, the consequences of the loss of privacy, and the measures taken to ensure this protection.

4.1 Flexibility of the system

The uploading of submissions will be one of the most frequent tasks done by a user. The submissions can come from a variety of sources (e.g. bLab, Submit), each source having its own file structure for exporting submissions. Furthermore, most course editions from bLab do not follow the same folder structure. These differences across different platforms and courses need to be accounted for when designing the system. The initial idea was to have a pre-set folder structure for each course/platform, but this negatively affects the user experience, as the user will have to select the source each time he/she wants to upload new submissions to the system. This idea quickly evolved into the current solution: creating a recursive upload, that can handle files dynamically. This allows the user to upload assignments from different platforms and courses with ease without having to select the source. The system automatically detects the file structure of the assignment and stores it into the system, while also extracting the relevant information, like student number, assignment name, or file extension. This was achieved using the decorator design pattern (4.2.3) [20] to dynamically decorate a Folder object with either assignments or submissions.

Another important aspect in the design of our system, one which our client highly desired, was to reuse previously calculated results whenever possible. To accomplish this, each triple of two submissions and one algorithm is saved in the database, to be able to retrieve them later, without needing to recalculate them. When adding more submissions to a check, the system is designed to run the selected algorithms on the new pairs that get created, without running them again on the already calculated ones. When adding more algorithms, only the new ones will be run on all of the selected submissions. This highly cuts down the time required for running a check, as the system never compares the same submission twice.

4.2 Architecture

As there are no strict requirements for the system architecture, the development team was left with the difficult task of choosing the appropriate architecture that does not limit the future prospects of the system while not having a steep learning curve. The decision to use the Spring Framework was based on multiple factors like the popularity of the framework, previous experience using various frameworks, and the similarity to other Labrador systems (e.g. Queue, Submit). This section will provide a brief overview of the architectural design choices made when developing the system.

4.2.1 Algorithm design

The first area of focus was the performance of the plagiarism detection algorithms. As was identified in the survey (B.2) given to the stakeholders during the process of requirements engineering, one of the most common points of feedback was that the current tools used for detecting plagiarism took too much time to run the algorithms as is also discussed in 2.6.3. Because of this, during the design process, it was important to ensure that the performance of the algorithms will not be affected by the development decisions of the system. Therefore, the platform was designed to avoid hindering the running of the algorithms. This decision could

already greatly improve the total time required to perform a plagiarism check. However, this would offer an even greater improvement when deployed, as systems with a higher core count would benefit more from a multi-threaded workload. In addition, the system was designed in such a way as to allow for relatively easily modifying the code to use separate systems for running the algorithms and calculating the results. This change could greatly improve the performance of the plagiarism checks as even more checks could be performed concurrently.

4.2.2 Database design

The system uses an SQL database for storing the locally-kept data. In addition, it is integrated with LabraCORE, so it retrieves user-specific and course-specific information from the LabraCORE database. The communication between the server and database is accomplished through Spring Data JPA [21] and Hibernate [22]. The design of the local database can be found in Figure 1.

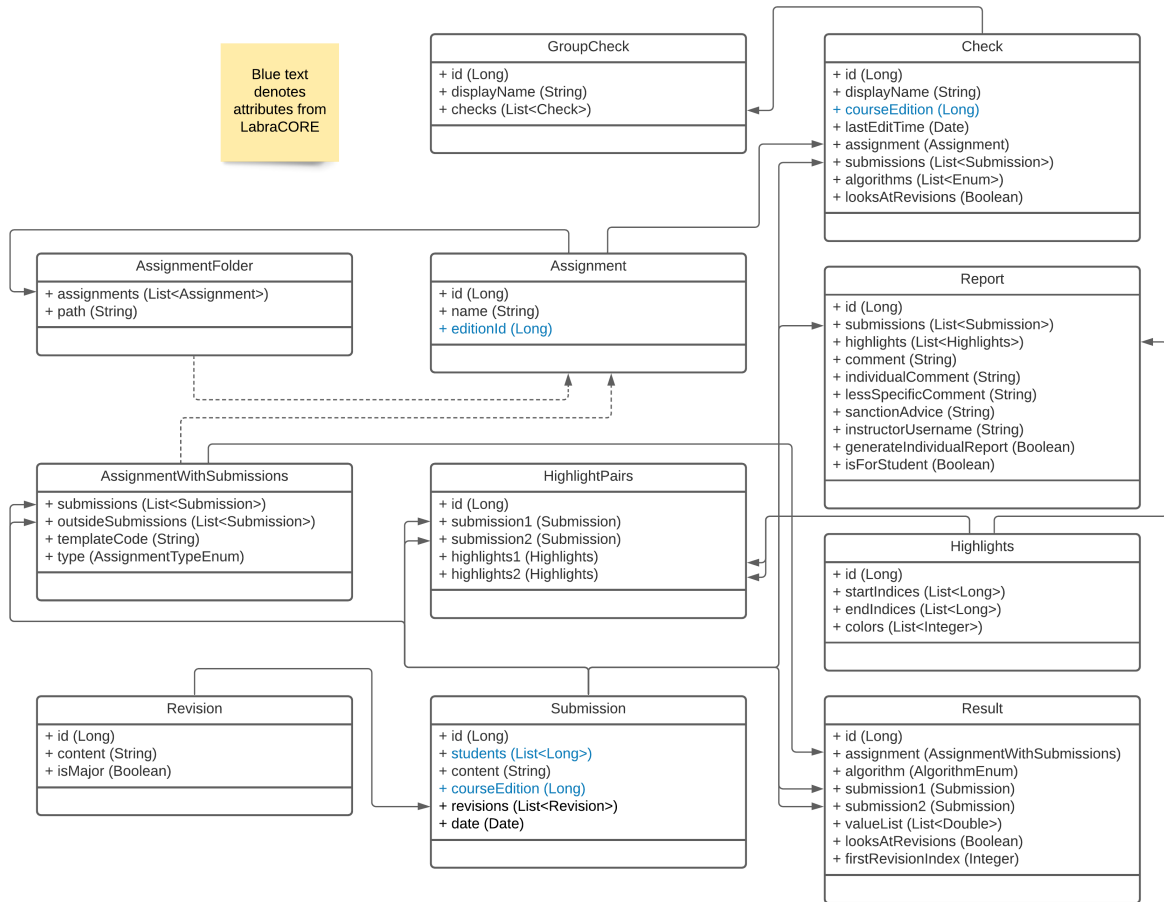


Figure 1: Schema of the database.

4.2.3 Design patterns

One of the hardest non-functional requirements of the system is to build a future-proof system that allows for easy expansion in the future. An important aspect of the system was the possibility to add algorithms later and also to be able to run different subsets of algorithms for each check. In order to facilitate this, the strategy pattern was used to design the algorithms. The strategy pattern is a behavioral design pattern that enables the changing of behavior at run-time. The algorithm interface defines all the methods needed by an algorithm. A newly added algorithm should simply implement the algorithm interface. This allows to easily select which algorithms are used on submissions at run-time and creates a template for adding new algorithms to the system later on. One downside of this is that the Algorithm interface might not completely encapsulate the required methods for an algorithm, and adding a new one might require implementing more than the methods covered in the interface. You can see a diagram of this design pattern in Figure 2.

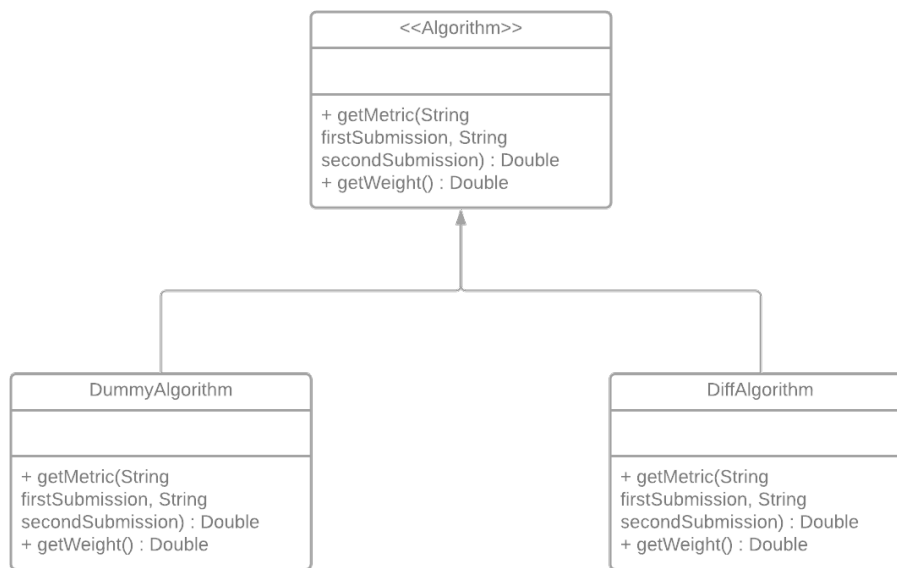


Figure 2: UML diagram of the strategy design pattern which was used to model the algorithms.

Another design challenge was posed by the structure of assignments imported from platforms such as Submit and blab. Often there is an assignment that contains further sub-assignments and so on until the actual submissions are reached. In order to model this behavior, the decorator pattern was used. It implements the abstract class `Assignment`, which can have two children: `AssignmentFolder` and `AssignmentFolderWithSubmissions`. Each of these children contains a list of `Assignments`. The folder is dynamically converted into this form and saved by the system. The advantage of this design pattern is its flexibility, because no matter the complexity of the file structure, it can be fully represented by the system's simple inheritance structure. On the other hand, one disadvantage of this solution is its maintainability because it creates a lot of similar code. You can see the diagram below, in Figure 3

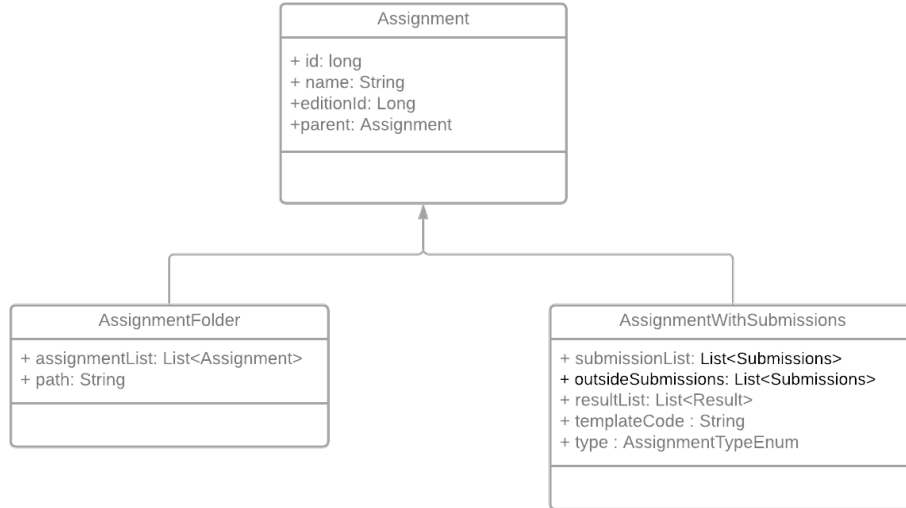


Figure 3: UML diagram of the decorator design pattern which was used to model the folder structure.

4.2.4 User interface design

The user interface was designed with our stakeholders in mind. The design was done with the help of [figma.com](https://www.figma.com) and it can be found in Appendix D. The focus was on having an easy-to-understand and easy-to-use application, which highlights the flexibility of our system. This design was approved by the client before the start of the implementation.

A good example of this would be the way the results page, which dynamically adds columns, based on the number of algorithms used. Furthermore, it concatenates multiple results under one row (Figure 25) so that the user can have an overview with the most suspect submissions. Then, if the user clicks on any row, a more detailed results page will be shown, comparing one specific submission with all the others (Figure 26).

One other example of a page that is easy to use, while also providing a lot of features, is the compare submission page, visible in Figure 27. This shows two (or more) submissions side by side, while also auto-highlighting the similarities between them. It also allows the user to manually add more submissions to the comparison and to highlight them manually. If the user chooses to report the current compared submissions, the highlighted submissions will be included as evidence in the report. The system ensures that multiple submissions could be compared in case of group fraud, so the page had to be designed with this in mind.

The Report Information page, that can be seen in Figure 29, needs to display all the relevant information that will be included in the report. Some of the information is retrieved from LabraCore (e.g student number, student name, course instructors, etc.), while some other information needs to be inputted by the user (e.g sanction advice, comment, etc.). The design of the page needed to consider all this, and combine on the same page, text boxes that contain editable text with text boxes that contain uneditable text.

One important aspect of the design process was making the application "fit in" with other already existing TU Delft applications. To do this, the TU Delft color scheme [23] was used as a reference throughout the application. For the design itself, such as buttons, how the pages are arranged, and other design choices, other applications that are being used at the TU Delft, such as Submit [24], Queue [25], and bLab [?] re used as inspiration.

To facilitate accessibility and transparency of the system, buttons have a clear label that describes their functionality. In some cases, icons will be used instead of labels, in such a way that the icons are descriptive enough for the user to deduce the buttons' function from it without

any additional text

4.2.5 Security

Since the system deals with sensitive user data, it is crucial to guarantee the security and of the application. To achieve this, the system applies the AAA principle: authentication, authorization, and accounting. In addition to simply determining if a user is permitted to access the system, the system must also provide user-specific access control using various user roles and course information from LabraCORE.

The current user roles that can access the system are TEACHER, TEACHER_RO, and HEAD_TA. These users only have access to the courses for which they have these roles in the LabraCORE database. Even though TEACHER and TEACHER_RO are different roles, they are both considered to have the authority of a teacher, which means they have access to the entire system. Comparatively, the HEAD_TA role has access to most of the system, but they are not allowed to view previously generated reports and export them.

The system uses Spring Security [26] to implement the security of the application. Using the LabraDoor library helps integrate the system with centralized identity providers by simplifying the process of connecting the application to an external authentication service. Although the development environment currently uses an in-memory database for retrieving user information, the system is already set up to use TU Delft SSO (single sign-on) authentication service after some changes in the security properties. By designing the system security to be adaptable, the product has achieved all of the security goals and requirements, as any further improvements like the deployment of the system are out of the scope of the project.

4.3 Ethical implications

The ultimate goal of the Plagiarism Detection System is to serve as a tool that ensures the fairness of the examinations. However, considering the fact that the system handles personal, thus (very sensitive) data, such as a student's number and submission, several ethical concerns may arise. In the upcoming sections, these concerns, as well as some preventative measures will be presented in-depth.

4.3.1 Protection of private data

According to Article 12 of the Universal Declaration of Human Rights [27], everyone has the right to protection against attacks on their privacy. Privacy is a fundamental human right and deserves to be treated as such. To this extent, the application makes no exception and is designed to secure its data carefully. Its means of authentication are done using LabraDoor, which is a library widely used within TU Delft's systems, providing authentication services. Even though reliability is one of the strengths of LabraDoor, any system failure might result in the data being compromised. However, having a shared implementation means that once the issues have been fixed from LabraDoor's side, the system will no longer require any further adjustments. Moreover, this is particularly useful, as it is no longer required to store information such as user names or passwords in the system's database. Although data security does not mean that the users have given their consent for their data to be used, this information is obtained from the university servers or other platforms used by the university, to which the students have given their consent. At the same time, the system uses only the minimal amount of information necessary to work and does not use more.

4.3.2 Social stigma as a result of identifiable data

As stated before, privacy is important as it protects people from harm. Identifiable data is data that can reveal the identity of a person if given access to it. An application used to track the

spread of Coronavirus in South Korea revealed embarrassing or ostracizing information about people that had been infected with the virus[28]. The same can happen in the case of students committing fraud, as data such as their student number, name, and course they are participating in is available to users. If a student commits fraud and somebody using the app (e.g. a head TA who has the necessary rights) makes this known for other students, then the person who was caught could face exclusion from social circles, embarrassment, or some form of abuse in more extreme cases.

Furthermore, even if the data stolen is not identifiable (the person cannot be directly linked to the data), the person's name or other details can still be determined if the person holding the data has other pieces of information. This is called indirect identification and is also an extrinsic loss of freedom. The victim could still fall prey to doxing [29] or other types of attacks.

Ultimately, it is the user's decision how their data or intellectual property is used, and their decision should be respected regardless of the risks involved, simply because the user is an autonomous being and their wishes should be respected.

4.3.3 Intellectual property

Code or text, the submission information that the application handles is the intellectual property of the student. Whether this property reflects a student's ideas, creative or innovative thoughts, or simply the understanding of the knowledge, this data needs to be protected. Similarly, even if the content submitted by the student is considered to be just partially modified from an existing source, this does not mean that this data should be protected any less. If a user of the plagiarism detection system uses an idea a student had in an essay and then profits from it in any way or uses it for personal gain, it is still theft. This is why access to any data such as submission and private information should be done as restrictively as possible and given to as few people as possible to protect the students.

4.3.4 Measures to ensure GDPR compliance and data protection

As mentioned before, LabraDoor provides the means of authentication. Thus, each user that is authenticated in the application is already provided with a role (e.g., student, teacher, TA, head TA), and the access to the application will be limited per role. For instance, a teacher will only be able to access data relevant to the courses he/she is responsible for, and a student will not be allowed to even access the system. A head TA can create a new check, but only the responsible teacher will be able to see the generated fraud reports. By doing so, it is ensured that a user sees only what is relevant when accessing the application, preventing the unnecessary spread of information.

Since the application stores information like students' submissions and student numbers, these have to be handled properly. Therefore, the personal information of students will not be stored longer than it is necessary to manipulate and analyze the data, according to the "storage limitation" [30] principle. Moreover, this data will be automatically deleted after a predefined amount of time.

5 Implementation of the plagiarism detection system

This chapter aims to provide an in-depth overview of the implementation of the system. Starting with section (5.1), it describes the software development process, including the approaches taken, the usage of GitLab, and overall communication processes. Next, the design choices are thoroughly explained in section (5.2). Lastly, the performance of the application in terms of both algorithms and data retrieval is discussed in section (5.3).

5.1 Development methodology

The successful development of the product requires not only an optimal implementation of code, but also effective communication, development processes, and usage of tools. These factors have been taken into careful consideration and have been integrated with the development process to improve the overall work quality and efficiency. This section will showcase the methodology of the development process that has contributed to the successful creation of the final product.

5.1.1 Processes

During the development process, a list of frameworks and guidelines had to be established. These rules were followed and monitored throughout the project to make sure that issues like task distribution, project planning, and code quality were being addressed.

Scrum framework

Throughout the project, the Scrum Framework [31] has been a central part of the development process. For this project, it had been decided that weekly sprints would be used, with each of them having a clearly defined purpose. The main goals of each sprint were selected at the beginning of the project by discussing the timeline of the product and creating intermediate deadlines. The sprint objectives can be found in the Wiki pages of the GitLab repository.

To ensure a good organization, each week a new scrum master was selected from the members of the team, who was responsible for creating the weekly issues and monitoring the progress of the sprint. At the beginning of a new sprint, the issues were distributed among the team members. In addition, all issues were given an estimated time to complete, to ensure that the workload was well divided and the sprint could be completed on time. At the end of each week, a sprint retrospective was made to discuss what went well, which were the main problems encountered, as well as possible improvements for the following sprints. Based on this retrospective and the feedback received from peers, the team members tried to adjust themselves to continuously improve the development process.

Testing

Since the beginning of the project, the system has been continuously tested to ensure that all of the functionality is working as intended. This approach has been selected to improve code quality and reduce the likelihood of unexpected failures. To guarantee that all code is being tested, each member of the development team was tasked with testing their contributions to the project. As many issues were discovered early due to the extensive testing, the overall code quality improved and the development team had greater confidence in the product.

5.1.2 Communication

As this project was done by a team of developers and it involved regular meetings with stakeholders, communication was an important factor in the successful completion of the project. Both communication within and outside the team involved regular meetings and status updates. Because this project was done remotely, all of the communication was done online.

Team. The main platform used by the team was Discord. Several channels were created regarding general questions and issues, planning of the upcoming meetings, and an overview of the progress. The team also held daily meetings, which were used to ensure that everyone was on track and had a clear understanding of the project status.

Teaching assistant. A Mattermost channel was created to stay in touch with the teaching assistant. This channel was used to ask questions to the teaching assistant, schedule meetings, and receive feedback. In addition to that, weekly meetings were held on Jitsi, where the team presented the results of each sprint and asked for advice in various implementation decisions or requested help regarding technical problems.

Client. Another Mattermost channel was dedicated to communication with the client. In this way, anything that was unclear or needed to be confirmed was discussed there, ensuring that the development team was application indeed meets the expectations of the client. Moreover, a weekly demo meeting was held to showcase the results of a sprint.

Coach. Communication with the coach was established via Mattermost and email for general questions regarding the project plan and the final report. One meeting was held in the second week to present and discuss the project plan and the requirements of the system. In addition, a midterm presentation was held with the coach, client, and teaching assistant via Zoom to present and receive feedback about the progress of the product.

5.1.3 GitLab

During the duration of the project, the development team made use of various tools which assisted in creating the product. The central and most used tool during this project is GitLab, as it was a crucial aspect of the project planning, team organization, and progress monitoring tasks. Therefore, this section will explain in more detail how it contributed to the overall development process.

First of all, the Kanban [32] board was set up such that there was a clear overview of open and closed issues, weekly and daily tasks. In addition, the issues were also separated based on if they were currently being done, awaiting reviews from other team members, or already completed.

Secondly, all GitLab issues were created based on an issue template (Figure 20), such that it contained a clear description, a definition of done, and a suggested way of resolving it. As the development process followed the Scrum Framework, each issue was assigned with a milestone corresponding to that specific sprint. The time-tracking feature was also used, such that it was possible to determine whether it was feasible to finish all of the planned features for a specific sprint. Moreover, using time estimates was especially useful to ensure that the workload was well-balanced and that all of the members of the development team were contributing to all aspects of the project.

Each issue was completed on a separate branch, so it was required to merge the branches once a feature was implemented. To merge a new feature, a merge request (Figure 21) had to be made following a template, which included the details of the issue and information on how to properly test it. At least two of the team members needed to approve a merge request, and that was done only after properly reviewing the code, testing it for sufficient code coverage, and checking the documentation. GitLab was particularly useful to leave comments on specific parts of the code, which were addressed before merging.

Lastly, the GitLab Wiki feature has been very helpful for organizational purposes. It includes all of the notes from the meetings, as well as the length of each meeting, as can be seen in figure 4. Additionally, the GitLab Wiki also stores the sprint plans, retrospectives, and a checklist with all of the system requirements. By doing so, any past information could be easily accessed using either the sprint number or the specific date.

Meeting notes

Week\Day	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1	19.04	20.04	21.04	22.04	23.04
Week 2	26.04	27.04	28.04	29.04	30.04
Week 3	03.05	04.05	05.05	06.05	07.05
Week 4	10.05	11.05	12.05	13.05	14.05
Week 5	17.05	18.05	19.05	20.05	21.05
Week 6	24.05	25.05	26.05	27.05	28.05
Week 7	31.05	01.06	02.06	03.06	04.06
Week 8	07.06	08.06	09.06	10.06	---
Week 9	14.06	---	---	---	---
Week 10	---	---	23.06 - 14:30 Final presentation	X	X

Figure 4: A table with the daily meeting notes we took from the GitLab Wiki page.

5.2 Building the system

The final system has been built based on many implementation choices that have been made due to the limitations of various tools, requirements of the system, or individual preferences. As there exist countless ways of implementing some functionality, this section will showcase the implementation details that define this project.

5.2.1 Spring framework

The central part of the application has been developed using the Spring Boot framework, which provides the basic infrastructure for developing a Java web application. This basic infrastructure comes in the form of modules, which take care of database transactions, security, API calls, and much more [33]. This allows the developers to focus on building the actual application rather than developing the necessary infrastructure. To develop this product, many core Spring components were used. The following subsections briefly describe the purpose of the components and how they were used in the implementation of the system.

Entities

The system uses entities to model the basic objects previously discussed in Section 4.2.2. In order to model the entity relationships, the system uses annotations to denote the relations between various entities following the database schema. The system contains entity relationships of various types - many-to-many, one-to-many, and one-to-one. All of the relationship types are represented by a bidirectional relationship where each of the two entities is aware of the other. However, to avoid issues with ownership, one side of the relationship is always marked as the owner.

One example of the entity relationships can be seen between the entities **Submission**

and `AssignmentWithSubmissions`. Since a submission is part of exactly one assignment but an assignment can have multiple submissions, this interaction is modeled as a one-to-many relationship. Then, as the goal of this product is related to submissions rather than assignments, it can be considered that the submission is the owner of this relationship. Therefore, the `AssignmentWithSubmissions.submissionList` attribute is considered as already mapped by the owning side of the relationship.

JPA repositories

Once an entity has been created, it needs to be persisted by storing it in the database. To simplify this process, the system uses Java Persistence API (JPA) repositories, which maps entities to database tables and adds support for performing database queries [34]. The system uses this functionality to perform complex database queries which retrieve entities based on the values of certain parameters. One such query can be seen in Figure 5, where the system checks if there exists a result entity with the specified attribute values. This type of query is commonly used in the implementation of the system, as they provide an effective and efficient way of retrieving and filtering entities.

```
boolean existsByAlgorithmEnumAndFirstSubmissionAndSecondSubmissionAndLooksAtRevisions(AlgorithmEnum a1,  
Submission s1, Submission s2, Boolean looksAtRevisions);
```

Figure 5: An automated database query for determining if a matching entity exists.

Controllers

All communication with the server is managed by the controller classes. The controllers contain the central logic of the application, as the implementation responds to an HTTP request in various ways like serving the user the Thymeleaf templates, manipulating the contents of the database, and creating new entities. As this project is made using a template engine, the implementation of controller classes is closely correlated to the user interface and the data it displays. Therefore, the implementation of the controllers impacts the quality and performance of various user interface features like the dynamic folder view or the aggregated results page.

Services

Similar to controller classes, services are responsible for implementing crucial server-side functionality. However, unlike controllers, they mainly consist of components that are not directly related to the user interface or data retrieval. Although most of the services perform tasks related to the features of the product, service classes also include the security implementation, which deals with managing the access control of the system.

One of the most notable service functionalities is the unzipping of an uploaded folder and correctly creating the internal assignment structure. This feature is implemented by two different services corresponding to the main platforms that the system supports - `UnzipSubmitService` and `UnzipWebLabService`. Each of these classes is responsible for validating the file structure, converting it to the internal assignment structure, and creating new assignments if necessary. In addition to retrieving the student submissions, the service must also detect whether revisions are present and extract the student numbers from assignment names.

5.2.2 Graphical User Interface

The graphical user interface (GUI) was build using Thymeleaf. Thymeleaf is a server-side Java template for building web applications. The main benefit of this framework is that the front-end

is not dependent on the back-end, and can be easily modified. It works by injecting values in the template on the server-side, and then sending the compiled HTML file. It also has a module for Spring, making them a great fit for one another. Furthermore, other systems currently available at TUDelft use Thymeleaf, so using it would keep it consistent across applications. For all the reasons stated above, we choose this framework as our front-end solution.

The GUI that the system implements can be seen in Appendix E. The implementation differs slightly from the design (which can be seen in Appendix F) in the places where improvements could have been made. One of the main differences between the design and implementation was the addition of the sidebar (seen in Figure 32). This allows the user to easily navigate the app, without having to go back to the first page (as our initial design was confusing).

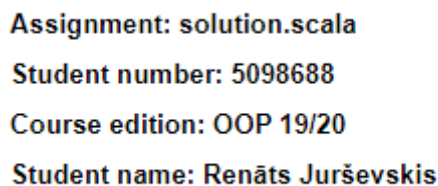
Another notable difference between design and implementation, is the new check page (seen in Figures 32 - 35). The design was using just one page for creating the new check (as seen in Figure 23), but the actual implementation of this page separates the process into stages: selecting submissions, selecting algorithms, selecting additional details, checking the selection. This highly improves the usability of the app, as the flow of creating a new check is more natural than it was in the design.

One other page that was changed in the implementation, was the results page (seen in Figure 36). This now also shows the current submissions being compared with all the others. This highly improves the user experience, because the user can actually see the contents, and doesn't just look at numbers.

As you can see, the GUI changed slightly between the design and implementation. However, all these changes were a conscious decision, because they improve the user experience in some way or another.

5.2.3 LabraCORE

One of the main functionalities of the system involves generating plagiarism reports using person information retrieved from an outside database. Therefore, a key area of the implementation process was integrating the system with the LabraCORE database which stores information about users and course editions that are used in the system. In addition to automatically filling in user and course information in the plagiarism reports, the system also uses LabraCORE to display course edition names, instructor names, and student names instead of the corresponding numeric identification values to improve the user experience Figure 6.



Assignment: solution.scala
Student number: 5098688
Course edition: OOP 19/20
Student name: Renāts Jurševskis

Figure 6: Submission information displaying data from LabraCORE.

All of this data is retrieved from the LabraCORE system using API (Application Programming Interface) [35] calls that return a data transfer object consisting of the necessary information. The application then processes this data and displays it where necessary. All of the important parameters for the LabraCORE setup have been set in a property file and can be easily modified when deploying the application. However, for the development environment, the system has been set to use a local copy of LabraCORE. More details about setting up the local instance of LabraCORE can be found in the `README.md` file.

5.2.4 Algorithms

One of the most central parts of the project is the algorithms used for detecting plagiarism. Although the main focus of the development process was not to implement advanced algorithms, the system still offers a selection of algorithms by including both a simplistic text-based algorithm and a more advanced token-based algorithm. The product implements the algorithms by using various libraries and adapting their results to reduce the amount of necessary work.

Diff algorithm

The Diff algorithm is a basic algorithm for detecting the common parts within the contents of two files. It analyses the contents and determines which sections are present in both submissions and which sections can only be found in one of the submissions. Then, this information is used to determine the common part of two files. The downside of this algorithm is that it detects just the very basic instances of fraud (i.e. Level 1 and Level 2 from Appendix A). In addition to detecting the locations of potential plagiarism, the Diff algorithm also supports the removal of an assignment template from a submission and detecting major revisions (i.e the percentage of lines changed is above a certain threshold).

JPlag algorithm

The system also implements the JPlag algorithm for detecting plagiarism [3]. First, the algorithm detects the submission type based on the file extension. Then, the contents of the submission are converted into tokens. If the file type is recognized by the JPlag algorithm, a special language-specific tokenization strategy is used, which considers both the language syntax and structure of the program. Currently, the list of natively supported file extensions includes **Java**, **Python**, **C++**, and **C#** files. However, if the assignment extension is not recognized, the tokenization is done by assuming that the submission contains plain text. Therefore, the JPlag algorithm can also be used for other languages but the returned results will not be as accurate and trustworthy. Using the generated tokens the algorithm finds similar sections of the two submissions and reports them as matching.

The JPlag algorithm returns a percentage value denoting the fraction of matching tokens. In addition, metrics such as the longest matching segment and total matching segments are also displayed. Since assignments can contain a template or boilerplate code, this algorithm also supports the automatic removal of the specified template. As JPlag performs tokenization and supports language-specific strategies, it can detect more advanced cases of fraud than the Diff algorithm. Therefore, this algorithm is recommended as the main algorithm for performing plagiarism detection.

5.3 Performance

To create a successful product, it was extremely important to complete the non-functional requirement of implementing a system that offers great performance. However, this non-functional requirement is difficult to measure as there does not exist a metric that would determine the performance of the system – it depends on many different variables and actions of the user. Therefore, a decision was made to focus on two areas for performance – plagiarism detection algorithm run-time and data retrieval from the database. For each of these metrics testing criteria were established to verify that the non-functional requirement has been completed.

5.3.1 Plagiarism detection algorithms

As mentioned in subsection 4.2.1, the code for running algorithms on submission was designed to support the use of multi-threading. To implement this design, the logic for running an

algorithm was separated from the creation of a check by extracting it to a separate method. All of these methods, including the method responsible for running a check, were implemented as asynchronous, which allows all of the submissions to be checked in parallel with the bonus of being able to use the application while a check is running.

Furthermore, any triple consisting of two submissions and one algorithm that has already been computed, will not be computed again but retrieved from the database. The method which handles running the submissions against each other is implemented in such a way that no redundant comparisons are made. This, paired up with the result retrievals, means that any check will require at most $n * (n - 1) / 2$ comparisons per algorithm. Therefore, if the time complexity of the algorithm itself is $O(m)$ for some value m , the total time complexity of running this algorithm with all submissions would be $O(n^2 * m)$.

5.3.2 Data retrieval

Another important aspect for improving the performance of the application is the process of data retrieval from the database. Since each database query is rather expensive, the database schema was created in such a way as to minimize the number of database queries. In addition, the implementation makes sure to store the results of algorithms in the database, so the plagiarism detection algorithms are not rerun as the results are already present in the database.

Another part of data retrieval includes making queries to the LabraCORE database. While the process of retrieving information from the local database is fast, using an outside database takes more time. Therefore, it is crucial for the performance of the application to minimize the number of times the outside database is accessed. At the moment the implementation transfers more data than optimal due to LabraCORE not offering API endpoints for the optimal database queries. Therefore, the performance of the API access could be further improved in the future by asking the LabraCORE development team to approve the addition of the new endpoint.

6 Product discussion and future enhancements

The goal of this chapter is to discuss and analyze the final product while also introducing some areas of improvement for the future. The first section will discuss the strengths and weaknesses of the product (6.1) while also justifying the decisions that led to prioritizing certain features over others. The second section (6.2) considers the future of the product and the potential improvements that could be made. And finally, the last section (6.3.1) presents an evaluation of the product.

6.1 Strengths and weaknesses of the system

The system contains both standout features that we would like to highlight, and parts that could still be improved in the future. Since the development process was resource-limited, prioritization had to be used to implement the most important features (see 3.1). This meant that some features were well polished, while other parts of the system were not implemented due to factors like the number of available resources, the short time frame of the project, issues not being identified during the design stage, and limitations due to the chosen tools.

Folder browser

When importing an assignment or exam the system generates a folder-like structure of assignments where each assignment consists of either another folder of assignments or a list of student submissions. Although this structure is great for storing an organized view of the assignments, it is not trivial to display this structure to the user due to a dynamic folder depth.

To solve this problem the folder browser was developed (Figure 7). This view automatically adapts to any folder structure regardless of the depth or number of sub-assignments. Using this folder browser the user can select the submissions used in a fraud check by ticking the corresponding checkboxes. If one of the assignment checkboxes is selected, the checkboxes of the child elements are also set to the same state (i.e all become selected or all become deselected). In addition, every assignment can be either minimized or maximized by clicking on the table row containing the assignment name. Since the internal structure of the assignments is created when a new file is uploaded, the folder browser can be used to display the assignments regardless of the import type. As the folder browser is defined as a fragment [36], it can be easily reused for various purposes without any code duplication, which was an advantageous design choice.

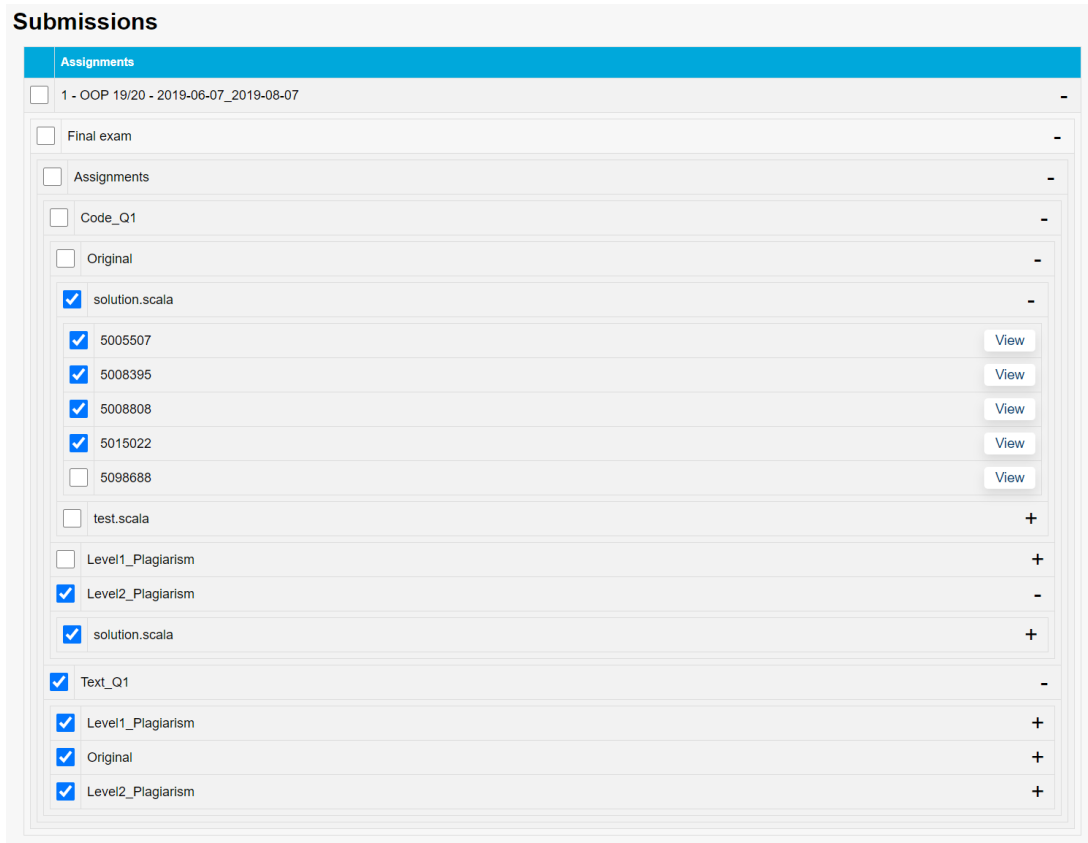


Figure 7: The folder browser view for selecting submissions.

Submission highlighting

Another standout aspect of the application is the submission highlighting feature. As highlighting the similar parts of the fraudulent submissions is a significant part of the plagiarism reporting pipeline, the system offers the users the ability to do so directly within the application (Figure 8) rather than using a third-party tool. By reducing the number of steps necessary to report a case of plagiarism, the system offers significant improvements to the user experience. In addition, this feature was one of the most requested features from the stakeholder survey from Appendix B.1.

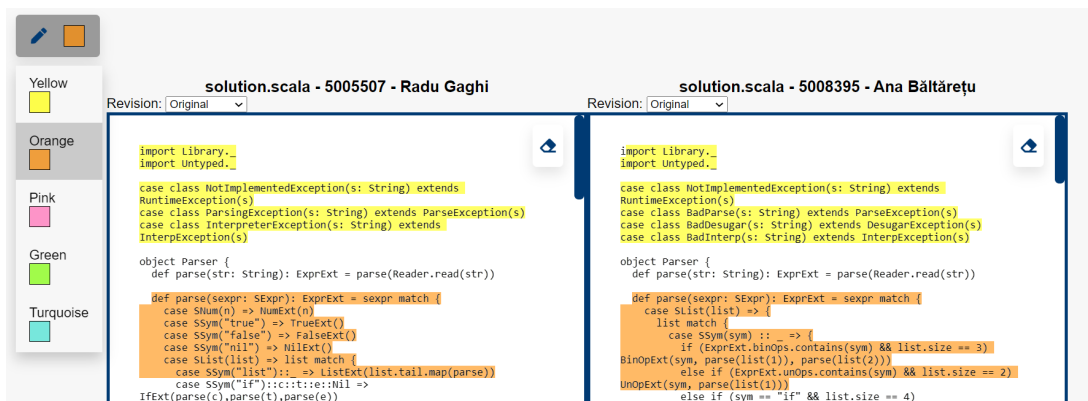


Figure 8: Parts that were manually highlighted by a user.

The one downside of the manual highlighting feature is that currently it does not support overlapping intervals, but, after a discussion with the client, the developers were assured that such a feature was as important as other features. The users most likely won't select two overlapping

intervals using different colors, since it does not make much sense to include the same part of the submission in two comparisons.

The submission highlighting feature not only allows the user to manually highlight the matching parts of multiple submissions but also provides the user with an automatically highlighted submission (Figure 9) based on various plagiarism detection algorithms. The user is then able to modify the automatic highlights as they see fit by adding new and removing existing sections. In addition, the system supports various colors that the user can use to more precisely identify the matching sections.

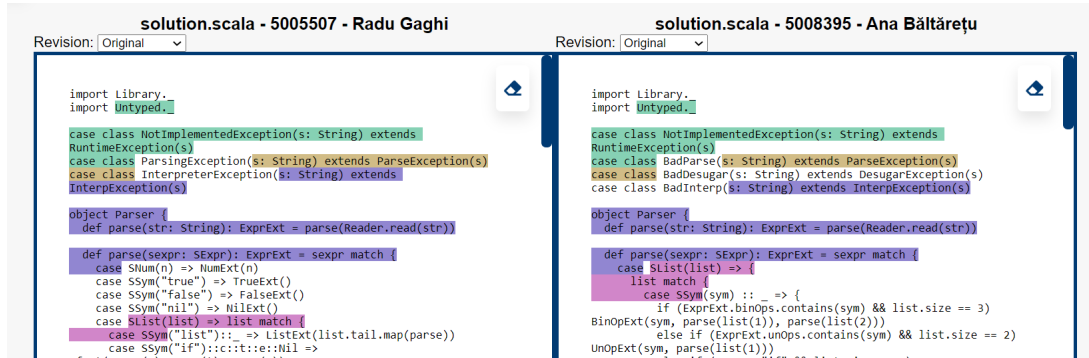


Figure 9: The automatically highlighted parts using the Diff algorithm.

Template removal

One other highly requested feature was the removal of "template files" from the submissions since for a lot of coding assignments most of the submissions are very similar. This is because the teaching staff provides the "template files" to the students to have some base code to write on, instead of having them do everything from scratch. The removal of this file from the submissions before running a check is very crucial because without this feature the scores would not have distinguishable differences, and most of the results would indicate a high match.



Figure 10: An example of an uploaded code template.

Revision

Even though the ability to import and use revisions when making a check was initially marked as a “could have”, due to high demand from the stakeholders, the developers decided to make it a “should have” and designed the application to support this feature. The final result allows the user to select whether or not they want to include revisions when making a check. Selecting that option means that a larger data set with multiple submissions from the same students is used, but doing so inevitably leads to an increase in the check’s running time.

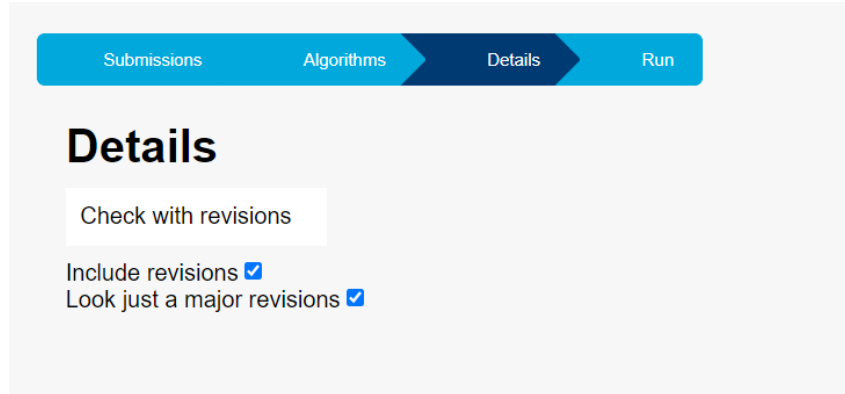
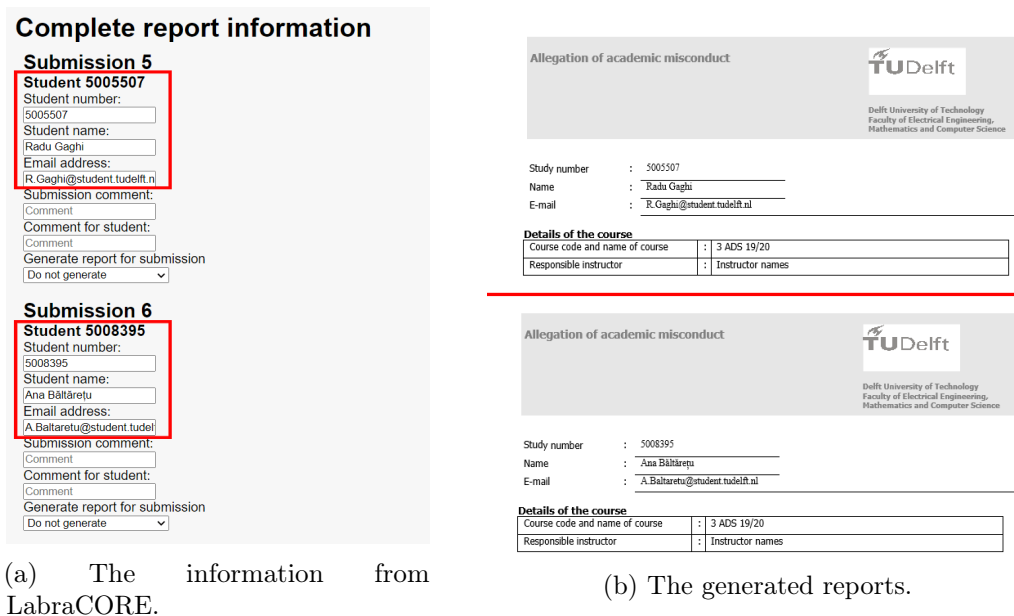


Figure 11: Including revisions in a check.

Automatically generating reports

To greatly decrease the time that was spent creating the fraud reports, the feature of automatically generating them based on student data from LabraCORE was developed. This feature (figure 12) allows the user to immediately see the information of the students after they decide to report the submissions, and this information is also stored in the generated DOCX and PDF files. This means that the teachers no longer need to spend countless hours manually filling in fraud reports.



(a) The information from LabraCORE.

(b) The generated reports.

Figure 12: Automatically generating reports.

Lack of algorithms

As the main focus of this project was to develop an easily extendable plagiarism detection system that improves the user experience compared to already existing solutions, the implementation of plagiarism detection algorithms was not the main focus. Therefore, the system currently only offers two algorithms - JPlag [3] and Diff. This can be considered as one of the main weaknesses of the system, as it fails to match other solutions in the number of algorithms. However, since it was established before the start of the project that the focus will not be on implementing algorithms, the current selection can be considered as acceptable. In addition, one of the main requirements of the system was for it to easily support the implementation of new algorithms. As the system has been developed with this in mind, the weakness of having a lack of plagiarism detection algorithms will not negatively impact the future development of the application.

User interface

Another weak aspect of the application is the relatively basic user interface. As mentioned before, the main focus of the project was to develop a basic application that could later be extended and improved. Because of this, the user interface, presented in 4.2.4, was not the central aspect of the product. This decision was also presented to the client and approved. Therefore, most of the work was focused on implementing new features, so the user interface designs were not improved as much. However, the product contains various UI features like the folder browser and the adaptive results page. The basic user interface can also be attributed to the development team not having much experience with creating a responsive and user-friendly UI. However, the application was developed in such a way that any future modifications to the user interface could be easily done with the help of people experienced in the field of UI / UX design.

6.2 Possible future improvements

Due to the short time in which this application was developed and a large number of requirements, there are many possible improvements and features which can be added in the future. One of the things that sit at the core of any plagiarism detection app are the algorithms, so adding more of these would bring a lot of value to our application. The system can easily support more algorithms so having more algorithms specialized in text or code and which could detect different types of fraud would greatly improve the chances of finding higher-level fraud cases. As all of them have different strengths and weaknesses, using more than one algorithm with multiple metrics on a check would be very beneficial and ensure the submission is analyzed from as many points of view as possible, further increasing the detection rate.

Furthermore, there are many quality of life improvements that could be made to improve the user experience and the degree of freedom users have when running checks. One such thing would be adding parameters for algorithms that have them, such that they can be configured according to the user's preference. Another feature that would improve the ease of use of the application would be the possibility of creating custom check templates that can be saved and used later. This could include the algorithm selection, whether or not to use revisions, and default values for parameters. Currently, the application can import assignments from two platforms: WebLab and Submit. It would greatly increase the versatility of our application and also its lifespan if users could create custom import structures to upload any kind of folders to the app. When selecting submissions, it is quite easy to select a large number of submissions, but selecting a small number takes much more time. As courses usually have between 300 and 500 students, looking up specific submissions is quite tedious and impractical. A search bar would solve this problem (assuming the user knows the names or student numbers of the submissions he is looking for) but it is quite hard to implement considering the dynamic nested table structure that is currently used when displaying submissions. Nevertheless, it would save a lot of time when running more specific checks.

While developing the app, it can be said that a large amount of time was spent on adding new functionality and much less time was spent on improving style and user-friendliness. With no exception, every page could be styled better and the overall flow of the application could more continuous, in the sense that a user with no idea about the application would have a very hard time figuring out how to use it. To this end, a FAQ or help page would make it much easier for inexperienced users to figure out how to use the application.

6.3 Product evaluation

In order to evaluate the product and make sure that it works as intended, a product evaluation has been performed. By analyzing both the user interface and performance of the system, valuable feedback can be received that could later be used to improve the quality of the application.

6.3.1 User interface evaluation

To make sure the design is up to standards, an expert was asked to test and evaluate our design. The expert was given a list of tasks and timed on how much it takes them to understand the system. The expert has been asked to give feedback based on his experience. This feedback could later be used to improve on areas that were not clear enough or impractical.

In the first UI test, presented in Appendix B.2, an expert was asked to go through the application and see how fast he can figure out how it works. The results were quite disappointing since the flow of the application was not clear to the user, and the confusing names on the buttons didn't help either. Some improvements were made such as adding names on all of the pages to clarify to the user "where they" are, the navigation buttons were renamed to more clear names, and they were added to most of the pages for easier navigation.

The second UI test, presented in Appendix B.3, clearly showed that there is still a lot of work left to be done on the details. For example, one of the experts that evaluated the application noticed some inconsistencies in the design. Another problem that was found was that some "clickable" elements are still not indicated properly and it is hard for a user to distinguish them from the elements that are just displaying information.

6.3.2 Performance evaluation

The overall performance of the application depends on many different variables like data retrieval, user interface responsiveness, and the performance of plagiarism detection algorithms. Therefore, it is not easy to measure and evaluate the performance of each of the contributing factors. System-wide tests were performed to evaluate the performance of the system, which can be seen in Appendix B.3. This user feedback was then later analyzed to determine the major areas of improvement.

The received feedback evaluates the performance as good. However, the system has mainly been tested on a small scale, using a limited amount of submissions. When tested with a larger submission count, the application is noticeably slower. These slower loading times can be explained due to the system needing to perform many database queries, as the data retrieval process significantly affects the performance of the system. However, these performance issues are not very alarming because the performance of the system has not been optimized yet, as the development team did not have the opportunity to test the system with a large number of submissions.

When implementing the plagiarism detection system, it was important to make sure that the selected algorithms offered sufficiently good performance by comparing the various options against each other. To measure the performance of various algorithms, tests were performed where the system was tested using the existing algorithms on a limited amount of submissions. Then, by using the runtime complexity of the comparison process, the expected runtime on a

larger set of submissions was computed. By comparing the run-time against systems currently used, the system performance could be better understood and it was determined whether the non-functional requirement for the system performance had been completed.

7 Conclusions and recommendations

Conclusions

The purpose of this report was to present the development process of building a Plagiarism Detection System for the Delft University of Technology that could serve as a replacement for the current technologies used to detect fraudulent activity in examinations.

The need for designing such a system was determined by several factors that affect both the teachers and the students. The most important factors considered were the time-consuming checks of hundreds of exam submissions that were done manually and the lack of products that could be easily integrated with the platforms available at TU Delft.

All in all, the new Plagiarism Detection System constitutes a solid foundation on which more functionality can be easily built upon. Since the development process is intended to be continued during the next months, further improvements will be added and ultimately, the application will be able to completely replace previously used systems.

It is expected that the system will have a profound impact on the education quality provided at TU Delft, as it will assure the students that their work is valued and acknowledged adequately. Moreover, it is envisioned that the rate of fraudulent examinations will drastically decrease, as the usage of such a tool will highly discourage students to plagiarize someone else's work. Not only the students would benefit, but also the teachers, whose time-consuming fraud checking work will be replaced by a completely automated system, giving them more time to focus on improving education instead.

Recommendations

Although our work on this project comes to an end, a piece of software is never finished and improvements are expected to be made. For future development, there are three recommendations.

Firstly, it is especially important to add more efficient algorithms that can detect fraud in both code and text. Even though implementing those was out of the scope of this project, the extensibility of the system was highly prioritized, thus any addition of algorithms can be easily integrated with the system. This would lead to more accurate results of fraud checking and would highly increase the chance of catching students copying other's work.

Secondly, since the system handles sensitive data, such as student numbers, names, and the actual submissions, which represent the intellectual property of the student, this data must be handled accordingly. Currently, this information is automatically deleted after a set amount of time, after which the information is no longer required. Thus, it is recommended that at any further stage of development, if new data is being used, then GDPR policies must be taken into account. It is the responsibility of the developers to take care of that.

Lastly, the usability of the application should be taken into account when making future changes. The process of creating a new plagiarism check should be divided into steps that can be easily followed, such that any inexperienced user could make use of the system. In that sense, it is suggested to add a help page for the users to clarify the flow of the application.

References

- [1] “Moscow method.” [Online]. Available: https://en.wikipedia.org/wiki/MoSCoW_method, May 2021.
- [2] “GDPR checklist for data controllers.” [Online]. Available: <https://gdpr.eu/checklist/>.
- [3] L. Prechelt, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with JPlag,” *Journal of Universal Computer Science*, vol. 8, pp. 1016–1038, 11 2002.
- [4] A. Bin-Habtoor and M. Zaher, “A survey on plagiarism detection systems,” *International Journal of Computer Theory and Engineering*, pp. 185–188, 01 2012.
- [5] J. Ryan, “Plagiarism, graduate education, and information security,” *IEEE Security Privacy*, vol. 5, pp. 62–65, 09 2007.
- [6] D. Chuda, P. Navrat, B. Kováčová, and P. Humay, “The issue of (software) plagiarism: A student view,” *IEEE Transactions on Education*, vol. 55, pp. 22–28, 02 2012.
- [7] C. Arwin and S. Tahaghoghi, “Plagiarism detection across programming languages,” vol. 48, pp. 277–286, 01 2006.
- [8] J. Faidhi and S. Robinson, “An empirical approach for detecting program similarity and plagiarism within a university programming environment,” *Computers Education*, vol. 11, no. 1, pp. 11–19, 1987.
- [9] S. Prasanth, R. Rajshree, and B. S. Balaji, “A survey on plagiarism detection,” *International Journal of Computer Applications*, vol. 86, 12 2013.
- [10] T. Kakkonen and M. Mozgovoy, “An evaluation of web plagiarism detection systems for student essays,” 01 2008.
- [11] “Blackboard app help.” [Online]. Available: https://help.blackboard.com/Blackboard_App.
- [12] “Moodle - open-source learning platform.” [Online]. Available: <https://moodle.org/>.
- [13] “Brightspace.” [Online]. Available: <https://brightspace.tudelft.nl>.
- [14] “The patriot act.” [Online]. Available: https://en.wikipedia.org/wiki/Patriot_Act.
- [15] J. Heather, “Turnitoff: Identifying and fixing a hole in current plagiarism detection software,” *Assessment Evaluation in Higher Education*, vol. 35, pp. 647–660, 10 2010.
- [16] The Plagiarism Resource Site, “Wcopyfind.” [Online]. Available: <https://plagiarism.bloomfieldmedia.com/software/wcopyfind/>.
- [17] “Plagiarism detection without limits.” [Online]. Available: <http://plagcheck.webtek.at/>.
- [18] M. J. Wise, “Detection of similarities in student programs: Yap’ing may be preferable to plague’ing,” *SIGCSE Bull.*, vol. 24, p. 268–271, Mar. 1992.
- [19] M. J. Wise, “Yap3: Improved detection of similarities in computer program and other texts,” *SIGCSE Bull.*, vol. 28, p. 130–134, Mar. 1996.
- [20] Refactoring Guru, “Decorator.” [Online]. Available: <https://refactoring.guru/design-patterns/decorator>.

- [21] Spring, “Accessing data with JPA.” [Online]. Available: <https://spring.io/guides/gs/accessing-data-jpa/>.
- [22] “Hibernate. everything data..” [Online]. Available: <https://hibernate.org/>.
- [23] TU Delft, “Colours.” [Online]. Available: <https://www.tudelft.nl/en/tu-delft-corporate-design/colours>.
- [24] “Welcome to submit.” [Online]. Available: <https://submit.tudelft.nl/>.
- [25] “Queue.” [Online]. Available: <https://queue.tudelft.nl/>.
- [26] “Spring security.” [Online]. Available: <https://spring.io/projects/spring-security>.
- [27] United Nations, “Universal declaration of human rights.” [Online]. Available: <https://www.un.org/en/about-us/universal-declaration-of-human-rights>, 12 1948.
- [28] The New York Times, “Major security flaws found in South Korea quarantine app.” [Online]. Available: <https://www.nytimes.com/2020/07/21/technology/korea-coronavirus-app-security.html>, 7 2020.
- [29] “Doxing.” [Online]. Available: <https://en.wikipedia.org/wiki/Doxing>, May 2021.
- [30] ICO, “GDPR.” [Online]. Available: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/storage-limitation/>.
- [31] “What is scrum?.” [Online]. Available: <https://www.scrum.org/resources/what-is-scrum>.
- [32] “Kanban board: Basics, features and how to use it.” [Online]. Available: <https://kanbanize.com/kanban-resources/getting-started/what-is-kanban-board>.
- [33] “Spring documentation.” [Online]. Available: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html>. Accessed: 2020-05-30.
- [34] “Introduction to the java persistence api.” [Online]. Available: <https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>, Jan 2013.
- [35] “Application programming interface.” [Online]. Available: <https://www.hubspire.com/resources/general/application-programming-interface/>, Mar 2021.
- [36] “Including template fragments.” [Online]. Available: <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#including-template-fragments>.

Appendices

A Level of code Plagiarism

Faidhi and Robinson have defined the following levels of code plagiarism in their study [8] :

- Level 1 - only comments and indentation is changed
- Level 2 – name identifiers are changed
- Level 3 – declaration of variables and functions is changed (e.g changing the names, shuffling the order of the instruction, etc.)
- Level 4 – variables and functions are changed (e.g two functions are merged into one)
- Level 5 – program loops are changed (e.g for loops are changed into while loops)
- Level 6 – control structures are changed (to something equivalent)

All levels include changes made in the levels above.

B Surveys

These are the surveys sent to experts and potential users of the system,

B.1 Requirements survey

This is the list of questions that was used to gather requirements from other stakeholders:

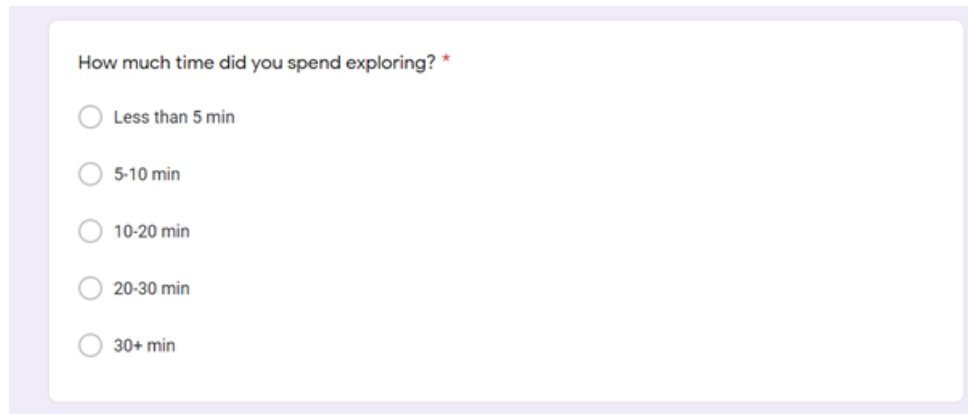
Questions for everyone
What is your current role? (TA / Teacher)
Have you used any other plagiarism detection software before?
Given some mock-ups, what would you change to improve the design?

People who have used a PDS	People who haven't used any PDS
What is the first thing you look for when comparing possibly fraudulent submissions?	What features would you like to see in a plagiarism detection system?
What features would you like to see in a plagiarism detection system that you do not have in the current one?	What would you find troublesome in a plagiarism detection system?
What features do you think are useful in the system you are currently using?	
In your opinion, what is the most important feature that should be improved from the previous system?	

Here is a link to the survey and the results: shorturl.at/prDM5.

B.2 User interface (UI) Survey 1

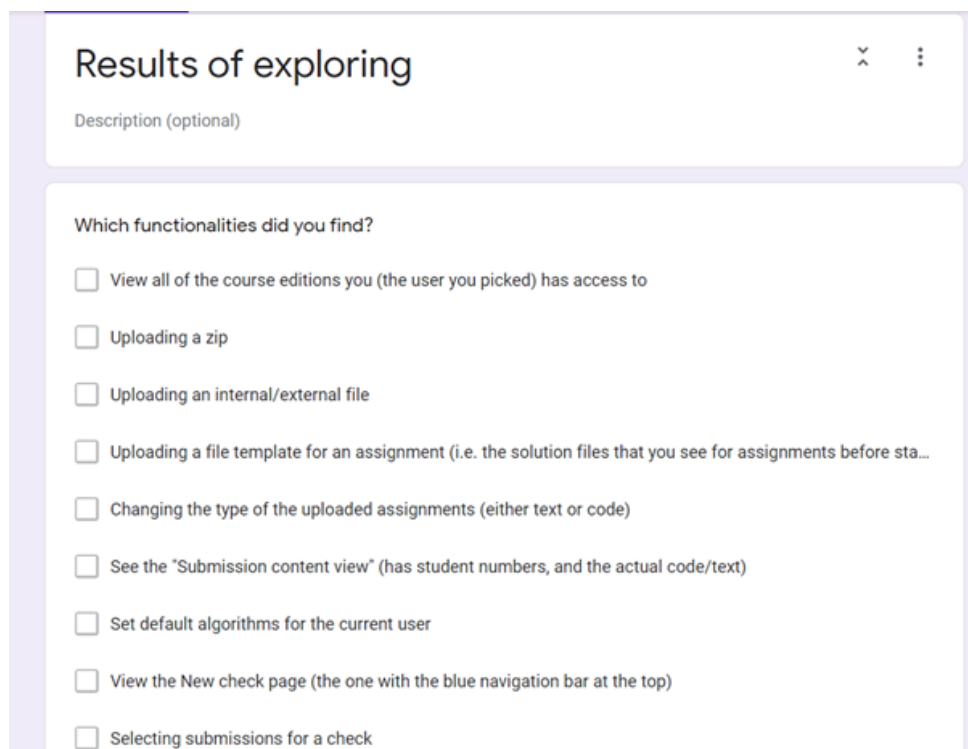
Here is a link to the survey and the results: shorturl.at/fkHQ6



How much time did you spend exploring? *

- ☐ Less than 5 min
- ☐ 5-10 min
- ☐ 10-20 min
- ☐ 20-30 min
- ☐ 30+ min

Figure 13: Page 1 of the first UI survey



Results of exploring

Description (optional)

Which functionalities did you find?

- ☐ View all of the course editions you (the user you picked) has access to
- ☐ Uploading a zip
- ☐ Uploading an internal/external file
- ☐ Uploading a file template for an assignment (i.e. the solution files that you see for assignments before sta...
- ☐ Changing the type of the uploaded assignments (either text or code)
- ☐ See the "Submission content view" (has student numbers, and the actual code/text)
- ☐ Set default algorithms for the current user
- ☐ View the New check page (the one with the blue navigation bar at the top)
- ☐ Selecting submissions for a check

The image shows a survey form with two sections, each containing a list of checkboxes. The first section has 13 items, and the second section has 6 items. The form is styled with a light purple border and a white background.

☐ Selecting submissions for a check

☐ Selecting algorithms for a check

☐ Select check details

☐ Review and run the check

☐ Realized that you can click on something without waiting for the entire check to finish running

☐ Saw the results of a check

☐ Added more algorithms to a check after running it

☐ Delete a check

☐ View the results of 1 student compared to all other students

☐ Sort based on different algorithm scores

☐ Filter based on id (aka the string that you see as the "submission")

☐ Filter based on minimum smart score

☐ Compare 2 submissions side-by-side

☐ Compare more than 2 submissions

☐ Highlight parts of submissions

☐ Remove highlights

☐ Report 2 submissions (get automatically filled in with info from labracore)

☐ View all of the reports

☐ Export zips with report and evidence

Figure 14: Page 2 of the second UI survey

Bugs and improvements

If you want to upload any pictures you can send them on Mattermost.

If you found any bugs you can describe them here, try to specify the URL from the top.

Long answer text

If you have any suggestions (UI design, making it more user friendly, buttons that you would rather put in other places) you can write them here.

Long answer text

Any other comments

Long answer text

Figure 15: Page 3 of the first UI survey

B.3 User interface (UI) Survey 2

Here is a link to the survey and the results: shorturl.at/kpyHY

Which functionalities did you find?

☐ View all of the course editions

☐ Upload a zip from WebLab

☐ Upload a zip from Submit

☐ Browse through the assignments to get to the list of submissions for a sub assignment

☐ Upload a file (external submission) in one of the folders

☐ Linking (internal) submissions from other assignments to the current assignment

☐ Upload a file template for an assignment

☐ Change the type of the uploaded assignments (either text or code)

☐ See the "Submission content view" (has student numbers, and the actual code/text)

☐ Set default algorithms for the current user

☐ View the "making a new check" page

☐ Selecting submissions for a check

☐ Selecting algorithms for a check

☐ Select check details (name of the check)

- ☐ Add more algorithms to a check after running it
- ☐ Select a subset of submissions to run a new check on
- ☐ Delete a check
- ☐ View the list of results for checks
- ☐ View the results of 1 student compared to all other students from a check
- ☐ Sort based on different algorithm scores
- ☐ Filter based on id
- ☐ Filter based on minimum smart score
- ☐ Compare 2 submissions side-by-side
- ☐ Compare more that 2 submissions
- ☐ Manually highlight parts of submissions
- ☐ Automatically highlight submissions
- ☐ Report 2 (or more) submissions
- ☐ View all of the reports
- ☐ Export zips with report and evidence

Figure 16: Page 1 of the second UI survey

How much time did you spend exploring?

☐ Less than 10 min
☐ 10-30 min
☐ 30min-1h
☐ More than 1h

Please rate your experience of using the app:

1 2 3 4 5 6 7 8 9 10
 Bad ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Good

How did you perceive the performance of the system?

Your answer _____

If you found any bugs you can describe them here, try to specify the URL from the top.

Your answer _____

Figure 17: Page 2 of the second UI survey

What parts of the UI were confusing?

Your answer

What parts did you like?

Your answer

If you have any ideas for future (after the SP ends) improvements you can write them here.

Your answer

Any other comments

Your answer

Figure 18: Page 3 of the second UI survey

C Contribution

Must haves		Names				
		Ana	Andra	Radu	Renats	Octav
1	Upload a batch of submissions (Weblab)					
2	Select type of submission					
3	Upload reference/online solutions					
4	Default algorithms selected					
5	Algorithms based on submission types					
6	Select multiple algorithms for a check					
7	Add new algorithms after the check is done					
8	Compare different course editions					
9	See metrics from algorithm results					
10	See list of matching submissions					
11	Filter the list of matching submissions					
12	Sort the list of matching submissions					
13	Sort on smart score					
14	Make and export report					
15	View reports					

Should haves		Names				
		Ana	Andra	Radu	Renats	Octav
1	Export reports as docx					
2	Make a less specific report for a student					
3	Multiple revisions					
4	Major revisions					
5	Upload from different platforms (Submit)					
6	Select your own default algorithms					
7	See results page					
8	Save results in database (to re-use them)					
9	Ignore template					
10	Get course/user information from labraCORE					
11	Side-by-side view of submissions					
12	Run more algorithms on a subset					
13	Multiple metris for algorithms					

Extra relevant features that aren't noticeable in the requirements		Names				
		Ana	Andra	Radu	Renats	Octav
1	View uploaded submissions / folders					
2	Have an extra algorithm implemented					

Could haves		Names				
		Ana	Andra	Radu	Renats	Octav
1	Manual highlight					
2	Auto-highlight					
3	See partial results while a check is running					
4	Make templates for running a check					
5	Make templates for importing a file structure					
6	Select subset of submissions and run a check					
7	See the highlights in side-by-side view					
8	Add more submissions to fraud report					
9	Customize parameters for algorithms					
10	Exclude empty submissions					
11	System detects file types					
12	Automatically group submissions in clusters					

Won't haves		Names				
		Ana	Andra	Radu	Renats	Octav
1	Mark submissions as "favorites"					
2	Write notes for submissions (side-by-side)					
3	View AST of a submission					
4	Compare only some methods from code					

Non-functional requirements		Names				
		Ana	Andra	Radu	Renats	Octav
1	System has a clear setup guide					
2	GDPR compliant (delete submissions)					
3	AAA					
4	Scalable and extendable					
5	Integration with WeBLab, LabraCore					
6	Clean, understandable UI					
7	Front-end easily modifiable					
8	Built in Java, Thymeleaf					
9	Login using LabraDoor					

Meaning of colors	
Done	
Some progress	
No progress	
What that person worked on	

Figure 19: The distribution of work based on the requirements

D Templates

Here are some of the templates we used throughout the project to keep our contributions consistent:

Write

Preview

B I ” </> 🔗 ☰ ☷ ⌵ 📄 🏠 ↶ ↷

Details
Please specify a description

Definition of Done
Please specify a definition of done

Possible solutions
Please specify a solution for the assignee to try

Markdown and quick actions are supported

Attach a file

Figure 20: The template for the issues

Write

Preview

B I ” </> ⌂ ☰ ☷ 🔍 ↺ ↻

```
# Merge Request  
Closes #*specify issue number*
```

```
# Checklist  
* [ ] Do you have sufficient test coverage (if applicable)?  
* [ ] Did you fix all style errors?  
* [ ] Did you achieve the definition of done?  
* [ ] Are you merging to the right branch?
```

```
# Details  
*Please specify any details regarding your merge request*
```

```
# How to test  
*Please specify how reviewers should test your functionality*  
![image](/uploads/83f2b01ce9951ceb2d88941969f62686/image.png)
```

Markdown and quick actions are supported

📎 Attach a file

Figure 21: The template for the merge requests

E Graphical User Interface (Design)

The Graphical User Interface (GUI) was designed with the user in mind, and it can be found below:



Figure 22: Login Page

New check

Name:

Include revisions: ☒

Select course edition

Select submissions

Select algorithms

Figure 23: Page that creates a new check

TU Delft | Check

Checks

Submissions



Find a check

Add check

Check name 1

Course name 1

Last edit date 1

[View reports](#)

[View details](#)

Check name 2

Course name 2

Last edit date 2

[View reports](#)

[View details](#)

Check name

Course name

Last edit date

[View reports](#)

[View details](#)

Check name

Course name

Last edit date

[View reports](#)

[View details](#)

Check name

Course name

Last edit date

[View reports](#)

[View details](#)

Check name

Course name

Last edit date

[View reports](#)

[View details](#)

Figure 24: Page that displays all the available checks

Check name



Submission	Smart score	Algorithm 1	Algorithm 2	Algorithm 3
Submission 1	a%	b%	c%	d%
Submission 2	a%	b%	c%	d%
Submission 3	a%	b%	c%	d%
Submission 4	a%	b%	c%	d%
Submission 5	a%	b%	c%	d%
Submission 6	a%	b%	c%	d%
Submission 7	a%	b%	c%	d%
Submission 8	a%	b%	c%	d%
Submission 9	a%	b%	c%	d%

Figure 25: Results page that shows the maximum similarity for each submission

Submission 1 matches with

Submission	Smart score	Algorithm 1	Algorithm 2	Algorithm 3
Submission 2	a%	b%	c%	d%
Submission 3	a%	b%	c%	d%
Submission 4	a%	b%	c%	d%
Submission 5	a%	b%	c%	d%
Submission 6	a%	b%	c%	d%
Submission 7	a%	b%	c%	d%
Submission 8	a%	b%	c%	d%
Submission 9	a%	b%	c%	d%

Figure 26: Results page that shows the maximum similarity between Submission 1 and all the other submissions

Compare Submissions

Report
Submissions



Add Comment



Submission 1

```
public class CustomEvent {  
    private String name;  
    private String description;  
  
    public CustomEvent() {  
    }  
    public CustomEvent(String name, String desc) {  
        this.name = name;  
        this.description = desc;  
    }  
    public String getName() {  
        return name;  
    }  
    public String getDescription() {  
        return description;  
    }  
    @Override  
    public String toString() {  
        return "CustomEvent{"  
            + ", name=" + name + ","  
            + ", description=" + description + "  
            + "}";  
    }  
}
```

Submission 2

```
public class Event {  
    private String desc;  
    private String name;  
  
    public Event(String desc, String name) {  
        this.name = name;  
        this.desc = desc;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getDescription() {  
        return desc;  
    }  
}
```

Figure 27: Page used for comparing and highlighting two submissions

Compare Submissions

Report
Submissions

 Add Comment 

Submission 1

```
public class CustomEvent {
    private String name;
    private String description;

    public CustomEvent() {
    }

    public CustomEvent(String name, String desc) {
        this.name = name;
        this.description = desc;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    @Override
    public String toString() {
        return "CustomEvent{"
            + ", name=" + name + "\n"
            + ", description=" + description + "\n"
            + "}";
    }
}
```

Submission 2

```
public class Event {
    private String desc;
    private String name;

    public Event(String desc, String name) {
        this.name = name;
        this.desc = desc;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return desc;
    }
}
```

Submission 3

```
public class Event {
    private String desc;
    private String name;

    public Event(String desc, String name) {
        this.name = name;
        this.desc = desc;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return desc;
    }
}
```

Figure 28: Page used for comparing and highlighting tree submissions

Student 1:	Student 2:				
Study number: 54891642761	Study number: 346256422748				
Name: Jimmy	Name: Fimmy				
E-mail: jimmy@student.tudelft.nl	E-mail: Fimmy@student.tudelft.nl				
<input checked="" type="checkbox"/> Generate report for Student 1	<input checked="" type="checkbox"/> Generate report for Student 2				
<input checked="" type="checkbox"/> Add individual comment	<input checked="" type="checkbox"/> Add individual comment				
Comment: <input type="text" value="Type your comment here"/>					
Sanction Advice: <input type="text" value="Type your advice here"/>					
Course Information: <table border="1"> <tr> <td>Course Code and name:</td> <td>CSE1234 Fraud Prevention</td> </tr> <tr> <td>Instructors:</td> <td>Lecturer name</td> </tr> </table>		Course Code and name:	CSE1234 Fraud Prevention	Instructors:	Lecturer name
Course Code and name:	CSE1234 Fraud Prevention				
Instructors:	Lecturer name				
<input type="checkbox"/> Instructor declares statement to be completed as true and correct					
<input type="button" value="Add to Batch"/>					

Figure 29: Page used for checking completing the report

Submission 1

```
public class CustomEvent {  
    private String name;  
    private String description;  
  
    public CustomEvent() {  
    }  
    public CustomEvent(String name, String desc) {  
        this.name = name;  
        this.description = desc;  
    }  
    public String getName() {  
        return name;  
    }  
    public String getDescription() {  
        return description;  
    }  
    @Override  
    public String toString() {  
        return "CustomEvent{"  
            + ", name=" + name + "\"  
            + ", description=" + description + "\"  
            + }";  
        }  
    }  
}
```

Figure 30: Page used for viewing one submission

F Graphical User Interface (Implementation)

The actual implementation of the GUI differs from the design slightly. You can check out the general look bellow:

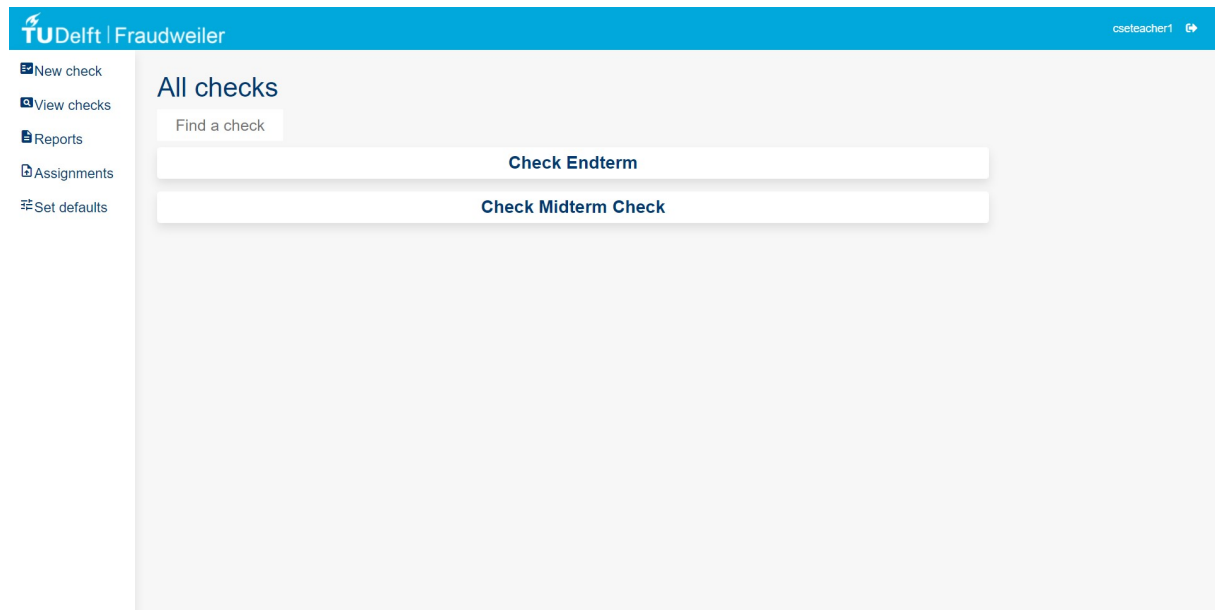


Figure 31: The main page of the application.

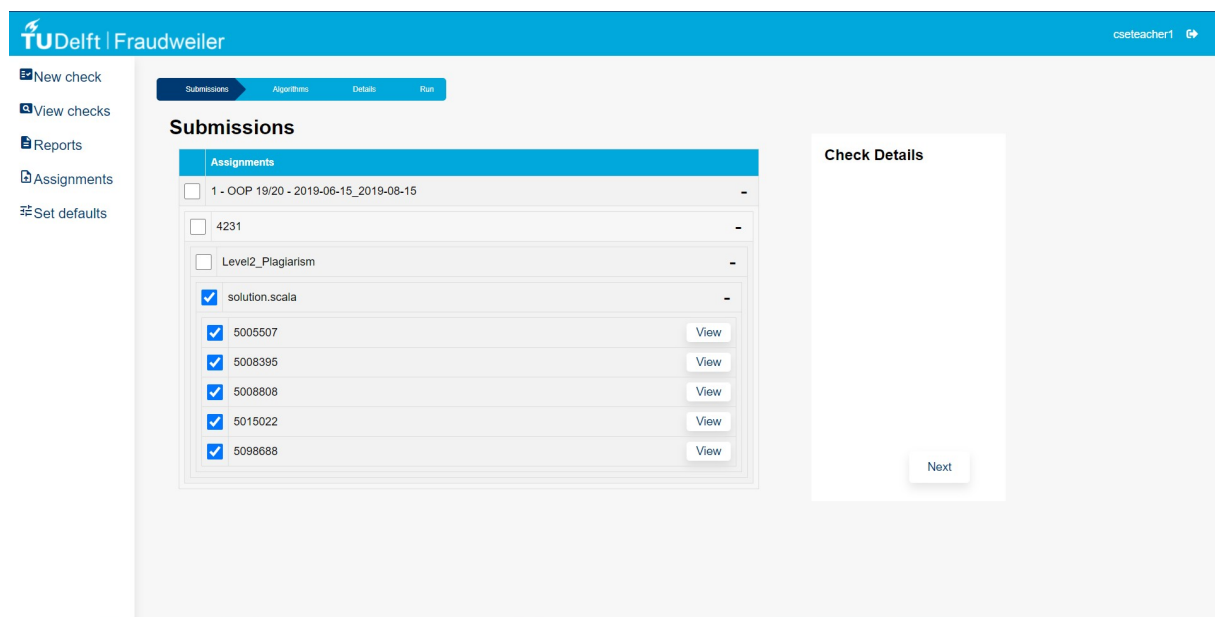


Figure 32: The page for selecting the submissions for a new check.

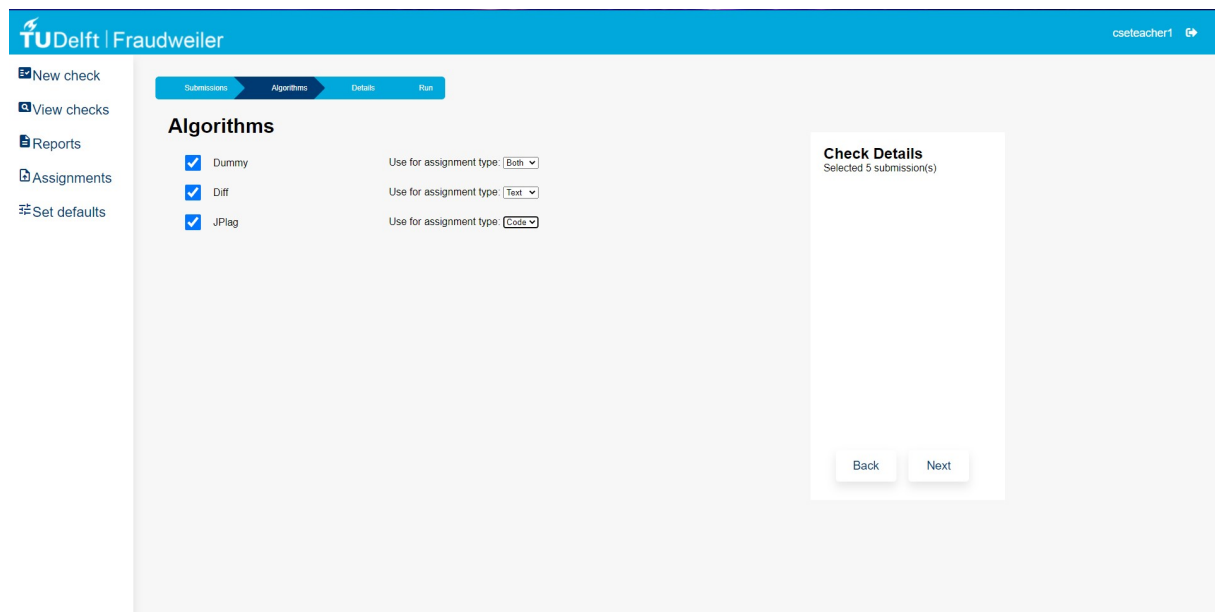


Figure 33: The page for selecting the algorithms for a new check.

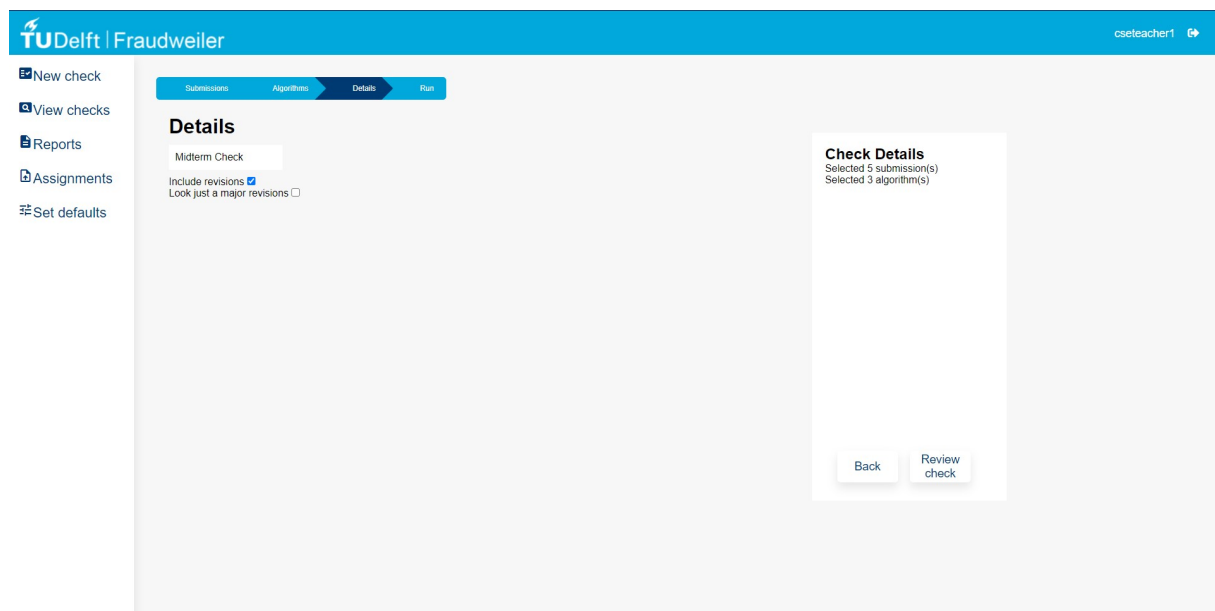


Figure 34: The page that let's the users name a check, and also select if they want to include revisions or not.

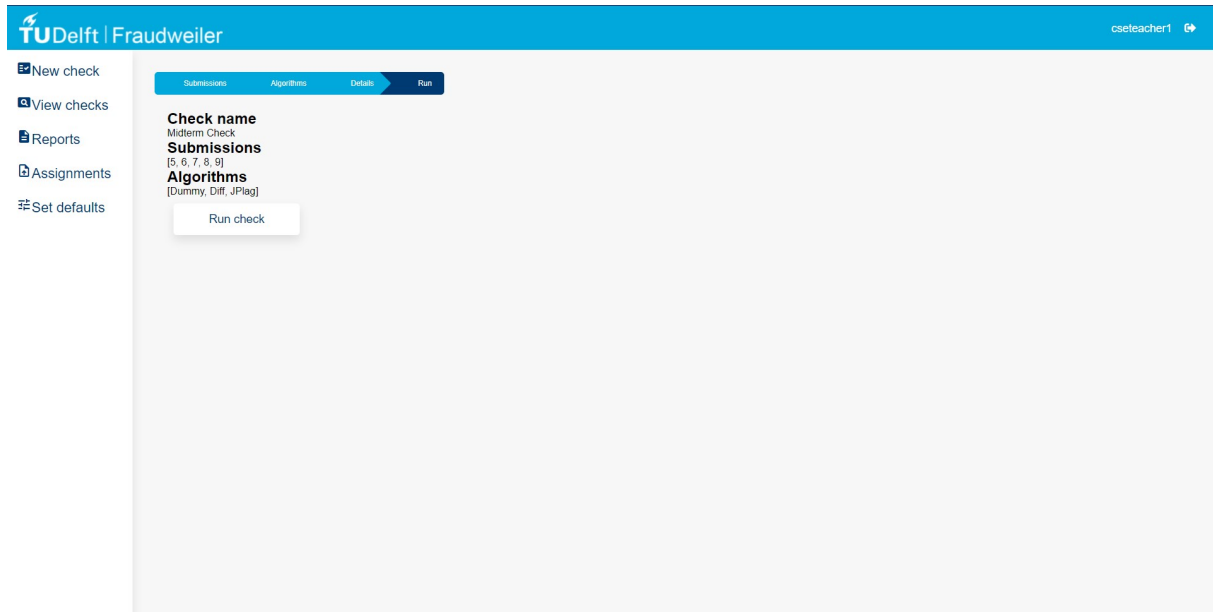


Figure 35: The page that displays all the information of a new check, before running.

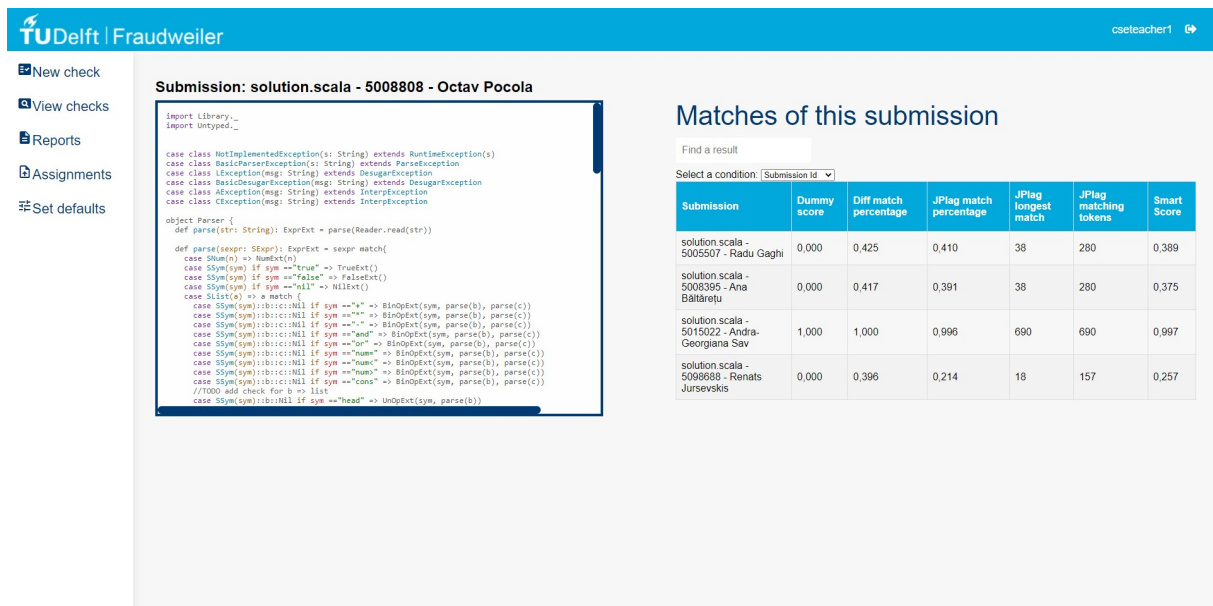


Figure 36: The page that displays the similarities between the submissions on the left and all other submissions.

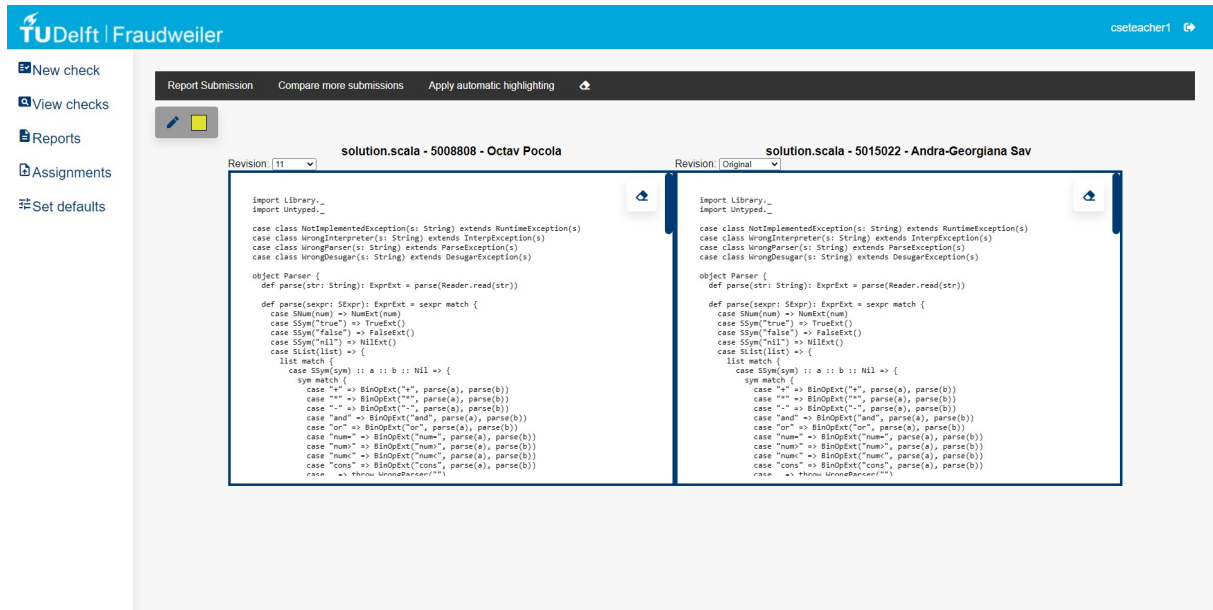


Figure 37: The comparison page with two submissions side by side

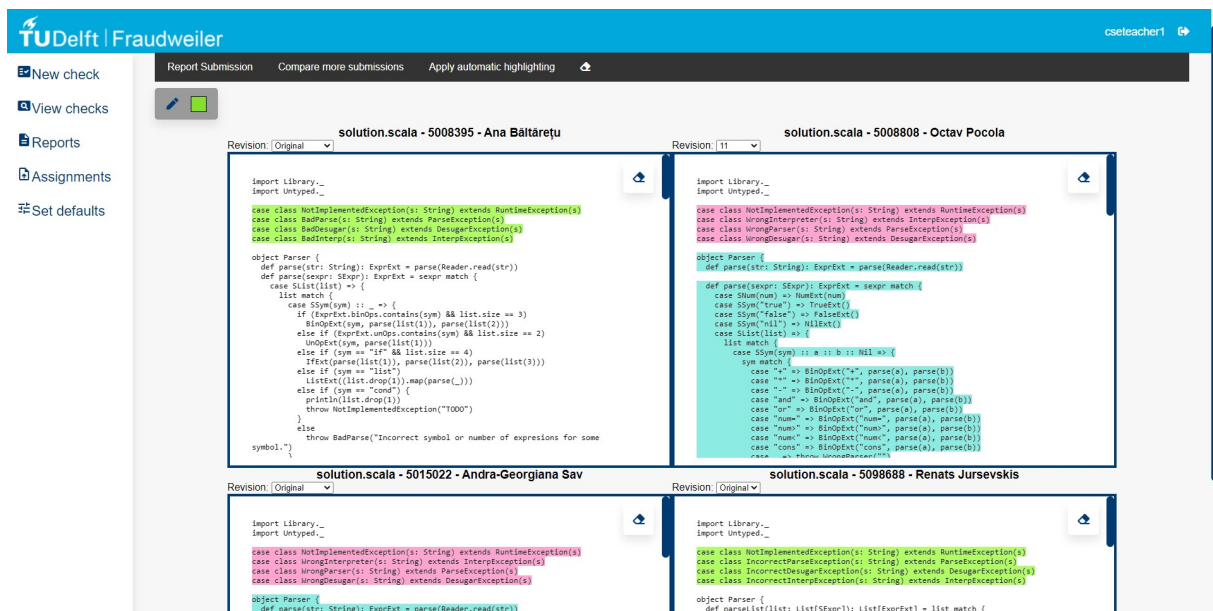


Figure 38: The comparison page with four submissions that are highlighted by the user.

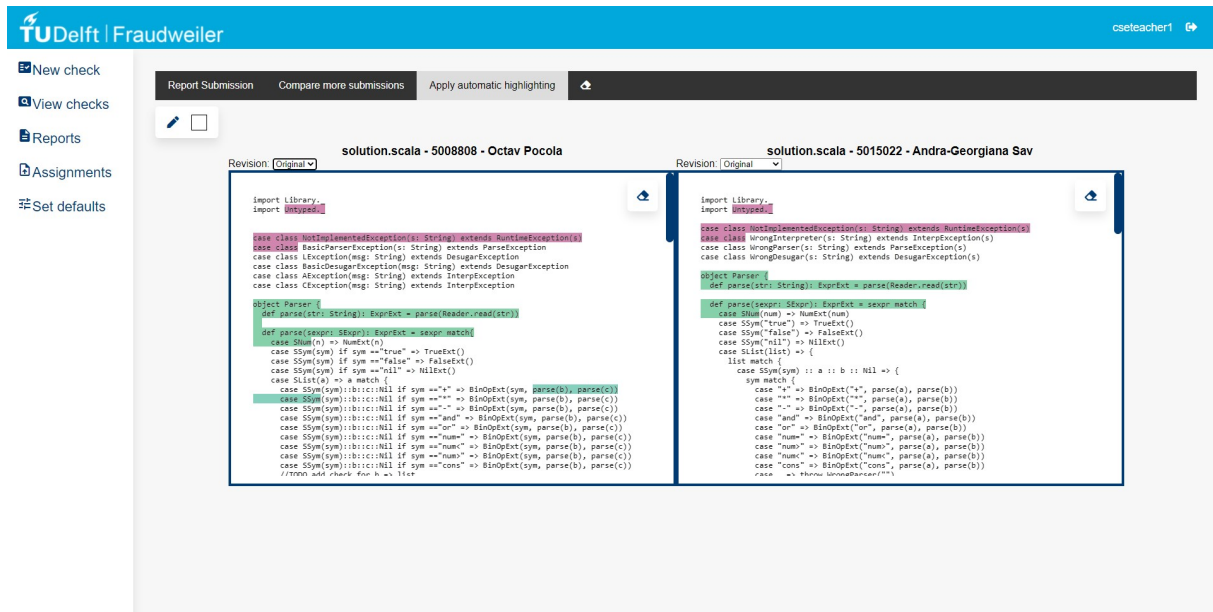


Figure 39: The comparison page with two submissions which are automatically highlighted by the system.

Figure 40: The report information field, where the user needs to specify information such as sanction advice or comments for the Board of Examiners. Most of the information is automatically retrieved

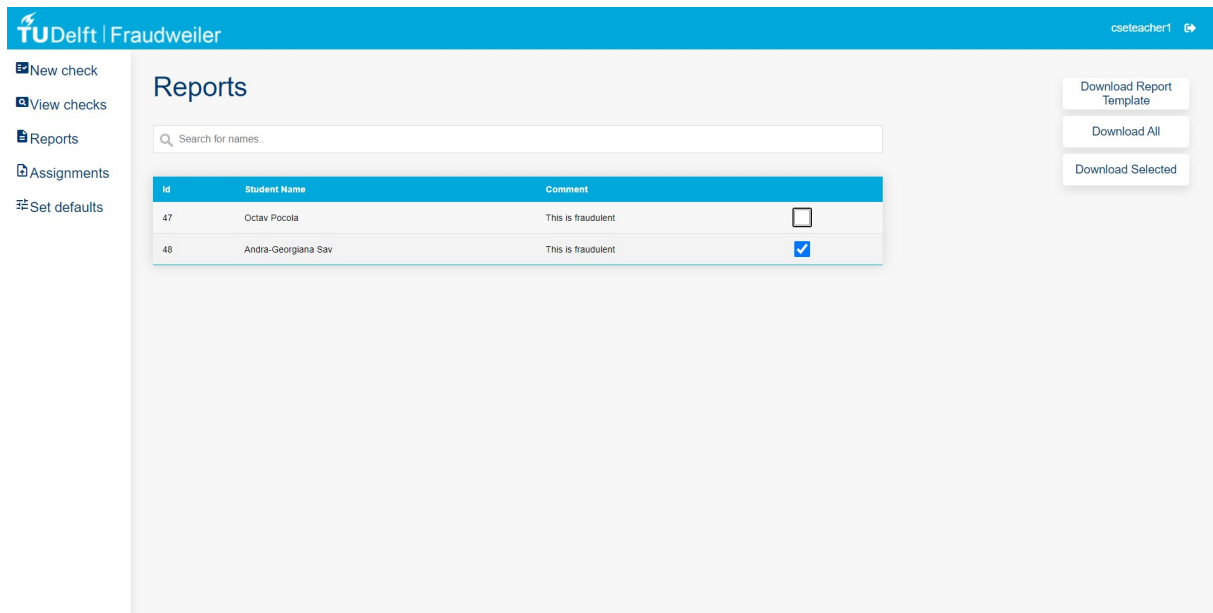


Figure 41: The reports page, showing all generated reports.

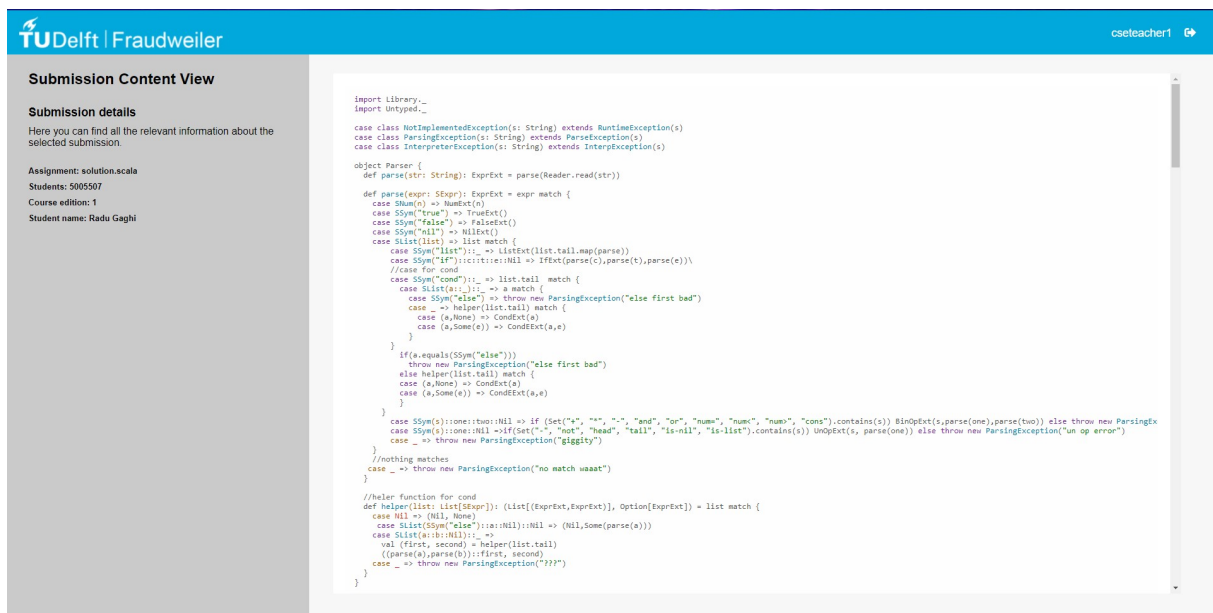


Figure 42: Page used for viewing one submission