Misconduct Assessment Tool

Stylianos Milisavljevic

4th Year Project Report
Software Engineering
School of Informatics
University of Edinburgh

2019

Abstract

Academic misconduct is a serious offence of any school's code of student conduct. The Misconduct Assessment Tool was developed to aid the Academic Misconduct Officer and other staff in taking a more thorough decision as it has severe consequences for the accused student. It supports the investigation of the less obvious cases that are suspected of committing source code academic misconduct, where multiple short segments of code are similar to other submissions. This project is the second development iteration and the goal is to extend the features of the system but also test and ensure reliable results. Improvements over the previous system include a number of user interface changes, report generation and the deployment of the system as a web service on the School's internal network. Additionally, the system was tested and the introduced solutions to issues found are analysed. Finally, additional features and improvements for future development cycles are discussed.

Acknowledgements

First, I would like to express my deep gratitude to my supervisor Kyriakos Kalorkoti, for his invaluable suggestions and constructive feedback. His willingness to give his time so generously has been very much appreciated. Throughout the work of this final year project I have received a great deal of support and assistance. Thank you!

I would like to acknowledge Yucheng Xie for his contribution to the system, giving me solid foundations to work on.

I would also like to thank my friends and flatmates for their friendship, making hard times look easy.

Finally, I wish to thank my family for their guidance, support and encouragement throughout my entire life. I wouldn't have got where I am today without you!

Table of Contents

Chapter 1

Introduction

Academic misconduct is a concept that can be traced back to the first examinations [41] and throughout time until the present day [20, 25]. A recent study in Germany found that 75% of university students acknowledged that they committed some form of academic misconduct during the last six months [46]. For Informatics students as most of their work consists of written source code, academic misconduct is usually committed by trying to pass off parts of their submitted code that were written by somebody else as work of their own.

Manually going through a group of submissions and trying to find signs of misconduct is inefficient and almost infeasible. Similarity detection systems thus were developed and attempt to find similarities between a collection of submissions [44, 17, 49]. Usually, they return a measure of similarity between each pair of submissions. This measure is then taken into account by the investigator so that further examination is made.

The Misconduct Assessment Tool is a unique system that uses the power other similarity detection tools to aid the investigation of the less obvious cases, where there is a suspicion of misconduct due to multiple small segments being detected as similar to other submissions.

The project work presented in this report is the second iteration working on this system, with the initial iteration being completed by Yucheng Xie [60]. The main goals of this development cycle are to expand the inherited system with new features like PDF report generation and user accounts, as well as improve the system by ensuring more reliable results.

The source code of the project is publicly available as a repository on GitHub at https://github.com/iamstelios/Misconduct-Assessment-Tool.

Chapter Overview

The project report is divided into the following chapters:

Chapter 2: Background

Explanation of the purpose and motivation behind the Misconduct Assessment Tool along with a review of related systems. Description of the inherited system from the previous iteration [60], along with its chosen underlying technologies. Definition of the goals set for this development cycle.

Chapter 3: Design and Implementation

Walk-through of the current system with reasoning behind the design choices made. The new features are showcased and some of their technical aspects are discussed in more detail.

Chapter 4: Testing

Evaluation of the inherited system. The issues found and their introduced solutions are analysed. Outline of the investigation regarding the effect of some parameters used by the similarity detection package.

Chapter 5: Conclusion

Wrap up of the report with some thoughts about the general project. Discussion of possible new features and improvements of the Misconduct Assessment Tool that can be achieved given more development time.

Chapter 2

Background

This chapter details the objective and motivation behind the Misconduct Assessment Tool. Furthermore, the system designed and inherited [60] is introduced. Finally, the work done in this iteration of the development is discussed.

2.1 The objective of the Misconduct Assessment Tool

Academic misconduct can be defined as any type of cheating that occurs in relation to a formal academic exercise that is for credit and it is a serious matter for any University. Students have to abide by the University's Code of Student Conduct. Otherwise, an investigation is made by the School's Academic Misconduct Officer and those involved face serious consequences [15].

The academic misconduct detection can be categorised into two main fields: natural language and source code [40]. What concerns the School of Informatics and the Misconduct Assessment Tool the most, is source code misconduct. Students can commit source code misconduct intentionally or unintentionally by not citing the sources of their work. It can be argued that some amount of editing can be considered an 'original' work, but academics agree that the sharing of source code, design and implementation, is perceived as academic dishonesty if the sources are not properly cited [21].

There are a number of techniques that can be used to obfuscate the copying of someone's work. Some of them are: [30], [58]

- Adding, modifying comments.
- Adding whitespace (e.g. spaces, empty lines).
- Renaming identifiers.
- Changing data types (e.g. `int` $\rightarrow$ `float`).
- Changing the order of operands in expressions (e.g. $13 + 37 \rightarrow 37 + 13$).
- Moving methods.

- Replacing expressions with semantically identical equivalents (e.g. `while(x<5){...}` → `while(x<=4){...}` where `x` in an integer).
- Externalising strings (e.g. retrieving strings from external files, usually used for easier localisation purposes).
- Extracting inner classes.
- Generating setters and getters.

Detecting misconduct over hundreds of submissions is almost infeasible and certainly inefficient when done manually.

A number of similarity detection systems have been built to analyse program source code and report similarities. An early occurrence that was reported in 1997 (Ottenstein [44]), took into account only the number of operators and operands, making it fairly ineffective. Some later developments include [18] and [22], where more metrics are included and resulting feature vector is considered in the investigation. However, these systems continued to perform poorly [57], which led to systems that use more advanced comparison algorithms and different optimisations. Some of the most widely used source code similarity detection systems today are *MOSS* [17] which is a free, closed-sourced online service and *JPlag* [49] which uses token-based detection and is an open-source Java package (Sections 2.3.2 & 2.5).

All of these similarity detection systems have comparable use. The user selects all of the submissions (files) and the system then checks each submission pairwise for similarities. The system then reports the similarity measure for each submission, indicating the matching similar submissions.

It is important to understand that these systems don't actually prove (or disprove) that academic misconduct has been committed. Some cases are obvious, e.g. exact copies of other submissions, however many cases are not that straight forward and false positives are possible. The person investigating the suspect files has to review whether the suspect files with higher similarity measure, are in fact committing misconduct or caused by some other factor. For example, *stacks* and *queues* can be implemented in a standard way, and students might translate the pseudo-code shown in lecture slides with small differences or similarity can occur by pure coincidence as some classes tend to be large, thus increasing those chances.

The aim of this Misconduct Assessment Tool is to test student submissions that are not obvious cases, but suspected of misconduct due to multiple small segments being similar to other submissions. With a large number of submissions, it is fairly likely that some segments will be similar by chance. Those segments can add up and the increased likelihood of similarity suggests that a more careful examination is needed. This tool is used to help in such situations, by *estimating the expected number of submissions with that exact combination of segments* (more details in Section 2.2). It is built to assist the decision-making process, it does not replace it. The Misconduct Assessment Tool also does not replace the currently used similarity detection systems (e.g. *MOSS* and *JPlag*), but complements them.

A typical use case is that a user first compares all of the submissions made by the stu-

dents, using a similarity detection system. The user then analyses the results and finds some suspicious submissions that are not clear of committing misconduct. The user then refers to Misconduct Assessment Tool where the suspect submission's segments that are similar to other submissions are selected to be further investigated.

2.2 Expectation estimate

As noted above, the core purpose of the Misconduct Assessment Tool is to estimate the expected number of submissions with the exact combination of some set segments that are present in a suspect submission. In order to calculate this measurement the following steps are made:

1. The individual probability of each segment occurring in the submission is estimated by $p_i = \frac{k}{N}$, where k is the number of submissions that contain a similar segment, N is the total number of submissions and i is the segment number. Note that $N-1$ is used in the denominator instead of N when the suspect file is included in the total number of submissions.

2. The joint probability of all of the segments appearing together in a single file (the suspect file), is the product of the individual probabilities $p_1 \times p_2 \times ... \times p_S$, where S is the total number of segments. Note that this calculation assumes that each individual probability is *independent* from the others.

3. Finally the estimated expected number of submissions with that exact combination of segments is the product of the joint probability with the total number of submissions without the suspect file $p_1 \times p_2 \times ... \times p_S \times N$. Again, note that $N-1$ is used instead of N when the suspect file is included in the total number of submissions.

The resulting estimation should be the number of students anticipated to have the exact combination of the selected segments with the suspect file. If the expected number is lower than 1, a case of misconduct is suggested. On the other hand, if the estimation is less than 1, it can suggest that the suspect submission is not expected to happen under pure chance, considering the total number of submissions.

It is worth emphasising that these results are not conclusive and should only be used as an aid to the investigation. The system relies on the assumptions that the individual probabilities of the segments have independence, as well as each of them is estimated well by the frequency calculated.

2.3 Previous Iteration

As noted in the Introduction (Chapter 1), this project is building upon the inherited system created by Xie Yucheng [60]. Xie implemented the core system with basic functionality having in mind that it will be then extended by subsequent developers.

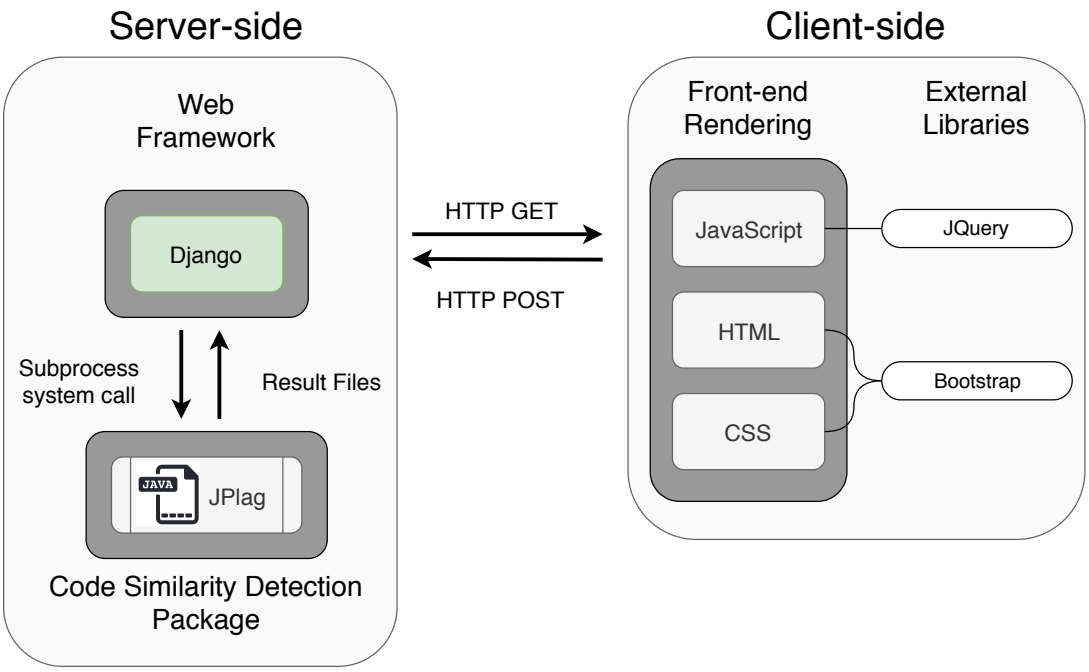


Figure 2.1: Architecture of the inherited system

A high-level diagram of the architecture can be found in Figure 2.1.

The system uses Django [12], an open-source web framework at its core. Django is used to process the web requests received by the users and acts as the back-end, server-side processor. JPlag is the similarity detection package used to detect similar segments to the given suspect submission segments, and thus the number of submissions that contain a similar segment (Section 2.2). When an investigation on the segments is initiated, Django executes as a subprocess JPlag, using its necessary parameters and then retrieves the results (HTML files) to be analysed and used to calculate the estimations. The files given to JPlag are the segments selected by the user to be suspicious, together with the rest of the submissions. The results files from JPlag include all of the pairwise similarity percentages between the segments and submissions.

The front-end content, which is delivered to the user's web client, uses the typical web stack of Javascript (programming), HTML (structure) and CSS (presentation). JavaScript's library JQuery [4] is used to navigate and manipulate HTML easily, as well as handle the asynchronous task of uploading the submissions using AJAX (Asynchronous JavaScript And XML). Bootstrap [45] is a front-end framework used to assist with the designing and fast prototyping of the interface as it contains design templates for most interface components (e.g. buttons, forms etc.).

The system was designed to be available as a web service. However, it wasn't deployed in the previous iteration. Before, users had to have a copy of the source files and run the system locally.

2.3.1 Web framework

The chosen web framework, Django uses the Model-View-Template (MVT) architectural pattern, inspired by Model-View-Controller (MVC) [39]. This allows for easy creation of database-driven websites. It uses an object-relational mapper to control and transfer data between Python (class representation) and a relational database ("Models"). Django also uses its web templating system to process HTTP requests and generate the web page, based on the request ("View"). Finally, its URL dispatcher acts as the "Controller".

Xie [60], based his choice of web framework on the work by [47]. The choice was between Django, Ruby on Rails and CakePHP. What gave the edge to Django was the fact that it performed better in the evaluation criteria than CakePHP and similar to Ruby on Rails. Django is written in Python which is a language used in some of the courses in the School of Informatics, thus easier to find future developers for later iterations because of their familiarity with it.

As it is not wise to change the web framework in this stage of development, no further investigation of web frameworks was done.

2.3.2 Code Similarity Detection Package

Xie [60] chose to implement the similarity detection part of the system with JPlag [49]. It used to be a closed-source web-based service, but now it is open-source and available at [3] as a Java application. The method of similarity detection is token-based, like SIM [29]. Other systems use different techniques: Sherlock [35] relies on digital signatures (the source code is mapped over a hashing algorithm), Checksims [32] uses Abstract syntax trees (AST) and some even feature-based neural networks [24]. MOSS (Measure of Software Similarity) [17] is close-source and based on the n-gram Fingerprinting algorithm with winnowing and is the main benchmark [19] used when developing other systems as it is still very competitive even though it was developed in 1994.

Xie, based his choice to use JPlag on the results of [30]. Xie argued that JPlag is open-source and supports the most languages out of the ones compared by [30].

Here, similarity detection packages are instead investigated on the basis of their performance in the similarity detection. It should be noted that because not all literature compares the same systems, here the most common and the ones that show promising results are included.

First, [30] concludes that JPlag and MOSS are amongst the top performing systems in the comparison tests and their results are fairly similar, while SIM behaves differently. It should be noted that [30] was published in 2011 and all of those systems have evolved.

Looking at some more recent publications, when Checksims [32] was tested against MOSS, it found every significant similarity identified by MOSS and even performed

Package Name	Methology	Open-source
MOSS	n-gram fingerprinting algorithm with winnowing	✗
JPlag	token-based	✓
Checksims	abstract syntax trees (AST)	✓
SIM	token-based	✓
Sherlock	digital signatures	✓
X9	static and dynamic source code analysis	✗

Table 2.1: Similarity Detection Packages examined

better than MOSS with small submissions. Because Misconduct Assessment Tool is concerned with segments smaller in size generally, the results suggest that Checksims is suitable from our system.

In 2018, the experimental results of X9 [48] (based on static and dynamic source code analysis), indicate that it outperforms the currently most widely used systems: JPlag, MOSS, Sherlock and SIM, especially when the source code is obfuscated.

A study [51] comparing JPlag, Sherlock and SIM, suggests that JPlag performed better in the Area Under Curve (AUC) metric when they drew the receiver operating characteristic (ROC) curves when varying the similarity threshold. However, SIM was the top performer with respect to false negatives and recall.

Given that some of the comparisons made do not state the parameters used by their experiments, systems might have not been tested using their optimal parameters, as suggested by [42]. When the systems are tested using their default settings, the differences between the nature of the submissions is not captured. For example, having a low sensitivity on small files will result in not finding any matches, and something analogous is implied for big files and high sensitivity. In [42], JPlag and SIM are tested using different sensitivity parameters and suggest that SIM ranks better than JPlag.

Comparing the languages supported by the mentioned packages, all of them support Java, Python and C++ which are the main programming languages used by courses in the School of Informatics. However, MOSS unlike the other packages, supports Haskell, Matlab and MIPS assembly which are languages that are being used in core courses [28, 37, 52] in the School.

Because not all literature compares all of the systems stated, no direct conclusion can be made. However, it can be said that JPlag does an adequate job in similarity detection being one of the top performing systems in most cases. A future iteration can use the results of this investigation and choose to introduce additional packages as an extension to the system, as suggested in Section 5.1. Because of the limited time involved with the project, this iteration prioritised other improvements of the system. As a result, the addition of other packages was left for another development cycle.

2.4 Aim of this development cycle

This development cycle aims to enhance the inherited system [60] with new features but also ensure that all of the problems found are fixed and the user receives reliable results. In this section the motivation behind the proposed enhancements and revisions is reported, while in Chapter 3 there is a more detailed analysis of the design and implementation.

The first enhancement is the support of generating a PDF report of the results. The inherited version of the system does not provide a method of exporting the data so that they can be shared. Because the results are to be used as evidence when assessing the case, it is important to have them available in a form that can be easily shared with the people doing the investigation and the student involved. The report should be easy to understand include some preamble explaining what is its purpose and the results with what they might suggest. Additionally, there should be options to add extra details in the report like notes, student matriculation number, course etc.

The second enhancement is to deploy the server inside a virtual server running in the School of Informatics internal network so that it can be accessed by the School Academic Misconduct Officer of Informatics Student Services, currently Kyriakos Kalorkoti who is also the project's supervisor. The deployed website allows the user to use and test the latest developments of the system without having to install or deploy a local version.

The third enhancement is the addition of user accounts. Creating a new user, logging in and out are abilities that the user should have. Also, user accounts can be used for storing user sessions and restrict access to the service by having some form of screening before activating accounts, but this is not to be implemented in this iteration.

The fourth enhancement is the creation of a log file. Any problems (e.g. runtime exceptions) are to be logged in the dedicated log file. Because the system is now deployed, the log file helps determine issues that occurred to users and were not reported.

Additionally, the inherited system has been thoroughly tested and a lot of revisions have been found to be necessary.

First, it was identified the inherited system was not detecting similarities for small segments. This is an important problem as the main use case of the system is focusing on numerous small segments in a suspect submission. The cause was found to be that the similarity detection package was not being executed with optimal parameters. After the current iteration this was fixed and the system is able to handle segments with a minimum of 5 lines of code.

Next, the inherited system was tested on different typical computer screen-sizes and it was found that the interface was not reacting correctly, as all of the components had fixed sizes. The improved system has to work on most screen-sizes and restructure the content dynamically for each.

In addition, a number of actions made the system produce incorrect or unexpected results as well as crash the system. After further investigation of their cause was pin-

pointed and a solution was provided. Some of the problems included actions that should not have been permitted (e.g. executing the similarity detection process without selecting any segments) and caused the system to crash. While other problems were not as severe, for example: adding additional empty lines of text in the code segments unintentionally (Chapter 4).

Finally, a number of interface changes and additions have been introduced to improve the User experience (UX). These changes should improve the use, accessibility and overall interaction with the system [55]. The latest UI patterns and components [50] are used to enhance user satisfaction. Some changes include syntax highlighting of the code and line numbering, the automatic detection and selection of the programming language for the submissions, as well as a number of warnings and tool-tips to guide the user.

2.5 Related Work

As suggested by Section 2.1, there is no other system that estimates the expected number of student submissions with that exact combination of segments. The vast majority of the systems focusing on source code similarity detection [17], [49], [59], [29] etc, provide the sole feature of giving an estimate of the similarity as a percentage between two submissions (Section 2.3.2).

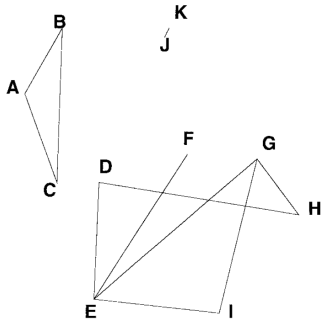


Figure 2.2: Kohonen self-organising feature map example [35]

Sherlock [35] visualises its results using a Kohonen self-organizing feature map [38]. Each node is a student submission. The lines connecting the nodes indicate significant similarities between their submissions and the shorter their length is, the stronger the similarity (Figure 2.2). Using this type of illustration can suggest groups of people collaborating, by visually detecting clusters of submissions that are connected.

In conclusion, none of these systems tackle the same use-case of Misconduct Assessment Tool.

Chapter 3

Design and Implementation

This chapter first lists the features of the inherited system from Xie Yucheng [60]. Then, the current system design and implementation choices are detailed.

3.1 Inherited system

As noted in the Introduction (Chapter 1), this project builds upon the system created by Xie and it is important to acknowledge his work as part of his master's thesis. The architecture of the system has been discussed in Section 2.3.

The inherited system provided the baseline for all of the future iterations. The features implemented by Xie include the uploading of the submissions and the suspect file to be analysed. Also, the selection of the suspect code segments was implemented and each segment is highlighted and referenced in a separate list of segments next to the submission's code. Then, the system prepares the selected segments in conjunction with all of the submissions to be processed by the code similarity detection package (Section 2.3.2). Finally, the results are interpreted and used to calculate the expectation estimate (Section 2.2). The inherited system is showcased in the following sections and is compared to the current implementation.

After extensive testing of this system (Chapter 4), a number of problems and usability issues were found, documented and repaired. These are discussed in more detail in the following sections, along with the new additions.

3.2 Overall Design

The current development iteration aimed to create a more robust system, fixing existing issues and introducing new features that aid and enhance the user experience. This development iteration modified the architecture of the new system (Figure 3.1) by using additional libraries and making use of an SQLite [33] database for managing user accounts.

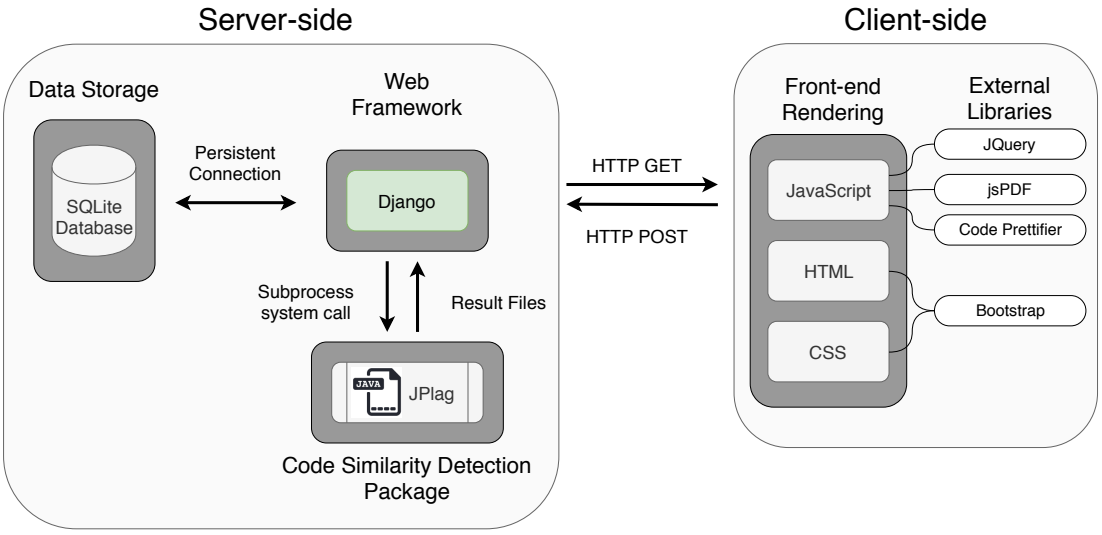


Figure 3.1: Architecture of the current system

3.3 Interface

A number of web interface changes and additions were made. Some of them were revising the current features making them more usable and improving clarity. Others were correcting usability issues, like the support of multiple desktop resolution renderings.

3.3.1 General

The instructions on each page have been revised to improve comprehension and provide more accurate and helpful information. The template format used in the entire website has four main components. These can be seen in Figure 3.2.

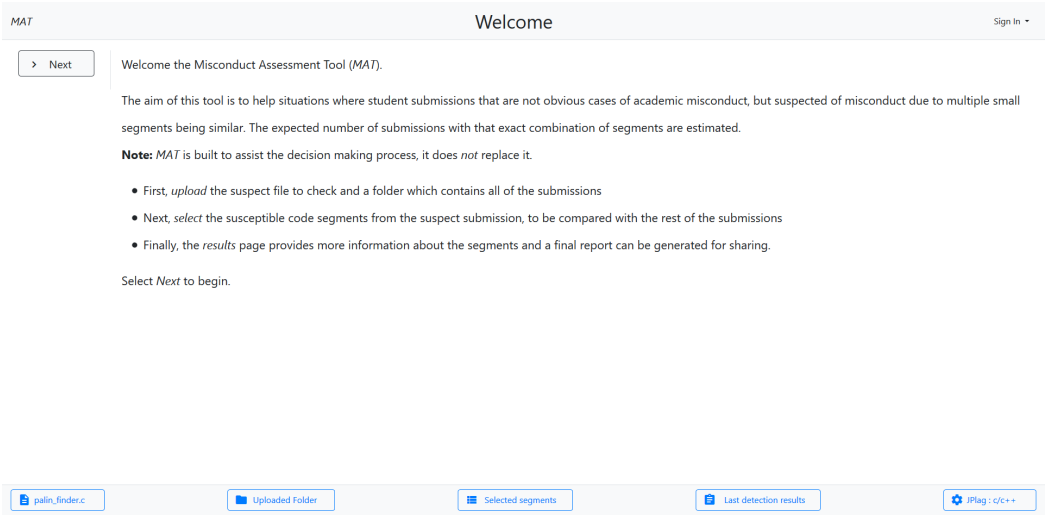


Figure 3.2: Welcome Page

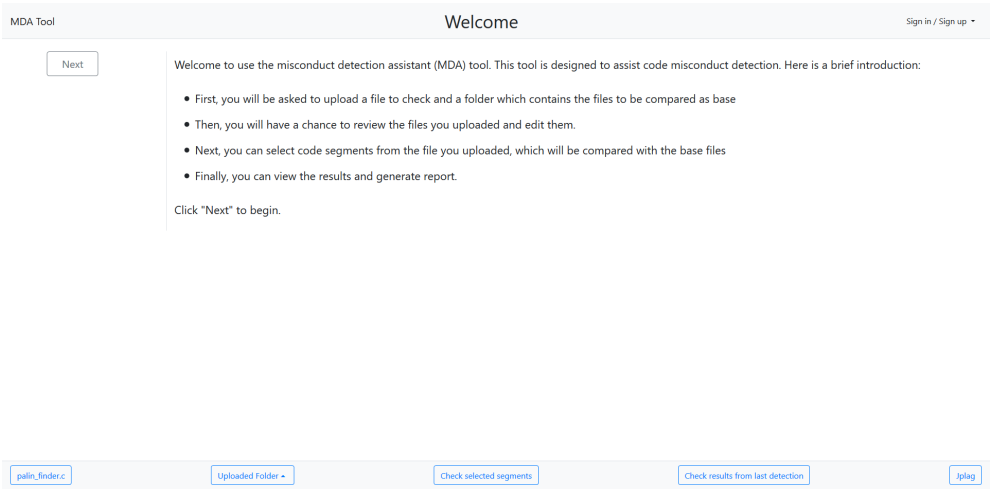


Figure 3.3: Previous system's Welcome Page

- The top title bar has the Misconduct Assessment Tool's initials, which is also a link to the welcome home page, the page's title, and the sign in drop-down button.
- The left navigation pane has the buttons for navigating through the system, and some extra feature buttons depending on the page.
- The bottom context bar has a link button to the uploaded suspect file, a link button to the uploaded folder page (Section 3.3.2), a link button to the segment selection page (Section 3.3.3), a link button to the last detection results (Section 3.3.5) and finally a button for similarity detection package settings tuning. These buttons are enabled depending on the state of the user's session.
- The content section, where all other information is included and varies with the page.

One of the most notable visual differences introduced in this development cycle is the addition of *icons* in each action button. The icons were chosen to convey the meaning of each action that its button performs. Not only are they visually pleasant and enhance the aesthetic appeal of the design, but also are *faster to recognise* [31]. Images have a tendency to last longer in long-term memory, as pieces of information are often fixed by the brain as visual images [53]. This translates in faster interactions with the interface. The text next to them was kept as it still helps overcome ambiguities and clarifies their meaning [31]. Results from a study [36] comparing menu items of text, icons and text-and-icons combined, suggests that when combined it results in the fewest number of incorrect selections by users.

Another essential change made, was the *restructuring* of elements and the support for multiple desktop screen sizes. The previous iteration used fixed positioning and shape sizes, rendering the system almost unusable in smaller screen sizes like the typical laptop resolutions of 1366×768 (Figures 3.4 & 3.5) and 1600×900. With 1366×768 being the most common desktop screen resolution worldwide holding 24% of the market [2] (as of March 2019).

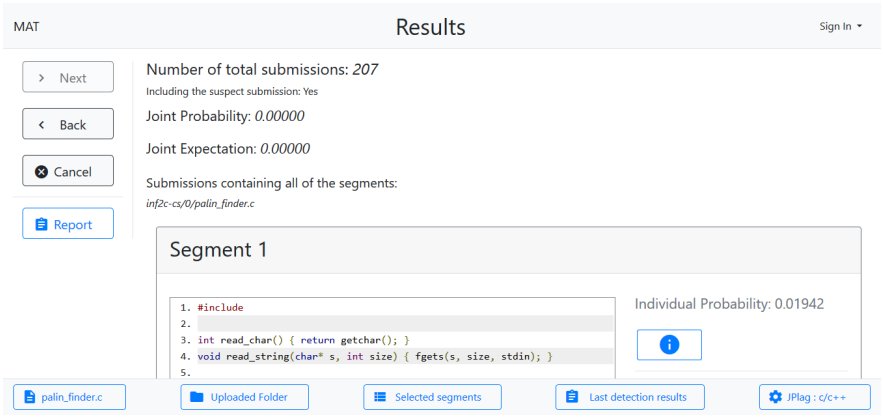


Figure 3.4: Results Page viewed on a 1366×768 laptop screen.

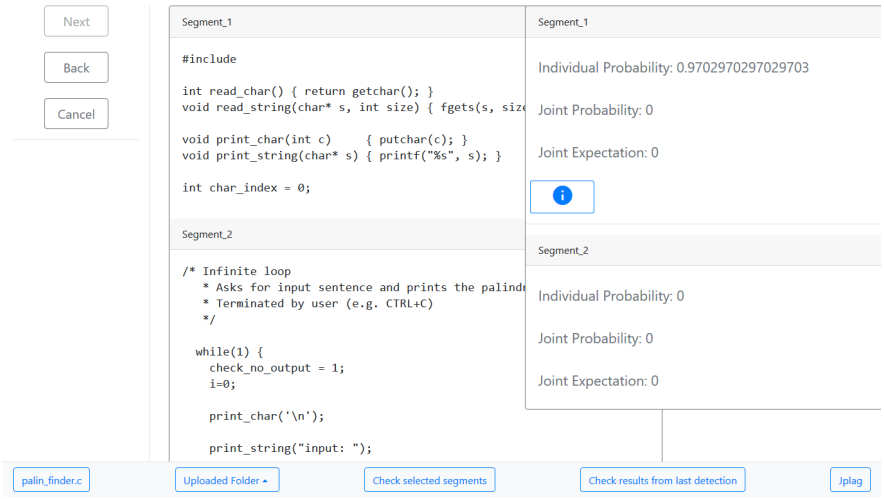


Figure 3.5: Previous system's Results Page viewed on a 1366×768 laptop screen.

This iteration utilises the *Bootstrap Grid System* [45] to dynamically layout and align the content of each page. This was accomplished using containers that used rows and columns to map out the design.

Bootstrap uses a 12 column system that has five responsive tiers (from extra small window width less than $576px$ to extra large, greater or equal to $1200px$). They were used to change the layout actively on different window sizes. For example, in the Segment Selection Page (Section 3.3.3), in screen sizes with width greater or equal to $1200px$ the content section is divided horizontally. The suspect file section is taking half the space (6 out of the 12 columns of the content section container), while the add segment button is $1/12^{th}$ and the segments section is $5/12^{th}$ of the total width (Figure 3.14). Otherwise, on window sizes less than $1200px$, the layout becomes vertical as both the suspect file section and segments sections cover all 12 columns of the content section container (Figure 3.6).

Finally, in this iteration the *Similarity Detection Package Settings* feature was implemented (Figure 3.7). It is a Bootstrap Modal dialogue, that is accessed by pressing the similarity detection package settings button on the bottom context bar. The settings

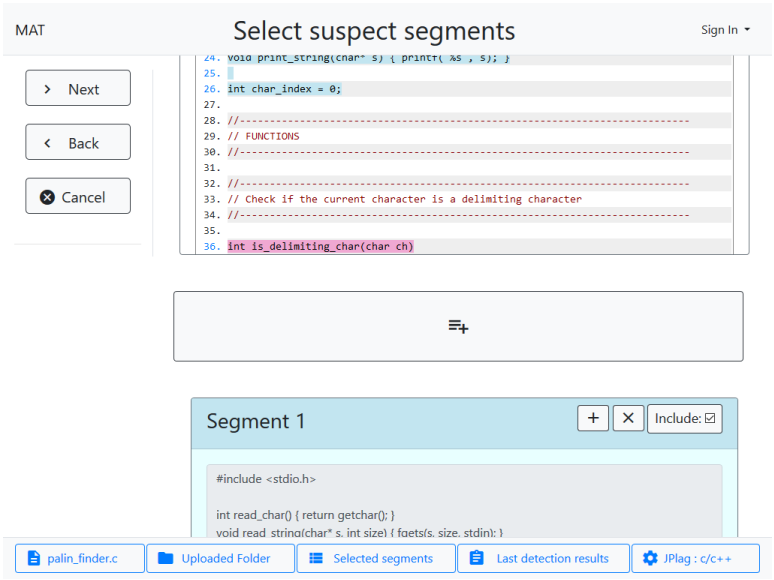


Figure 3.6: Select Segments Page viewed on a 1024×768 window size

were introduced so that the users have control of which detection package to use for the similarity detection between the suspect segments and the rest of the files. Note that in this iteration, no new detection packages are introduced. However, it is one of the future development plans of the system (Chapter 5). Also, the users can control the programming language and similarity detection threshold parameters used with the detection package.

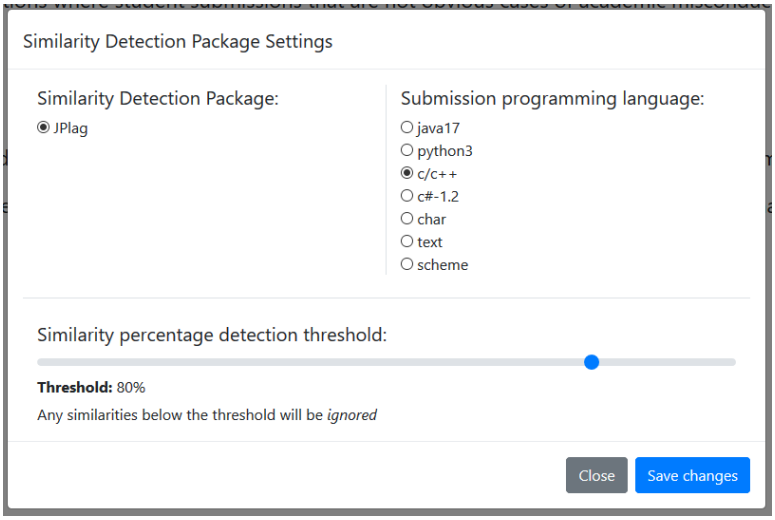


Figure 3.7: Similarity Detection Package Settings dialogue

In the previous iteration of the system, when a file was uploaded the programming language was determined from the extension of the suspect file and could not be modified. Now, when the suspect file is uploaded (Section 3.3.2), the programming language is automatically selected from the file extension. If the file extension is not usually associated with one of the supported languages, a dialogue (Figure 3.8) informs the user of the lack of native support for that file type. In this case, the file will be treated

as a plain text file, losing language specific detection features. The option to change the similarity detection package settings is given, showing the Similarity Detection Package Settings dialogue when selected. The user can always change programming language in which the submissions are being considered. It is important that the user chooses the appropriate language because each has to parsed differently (Chapter 2), i.e. programming languages have different syntax.

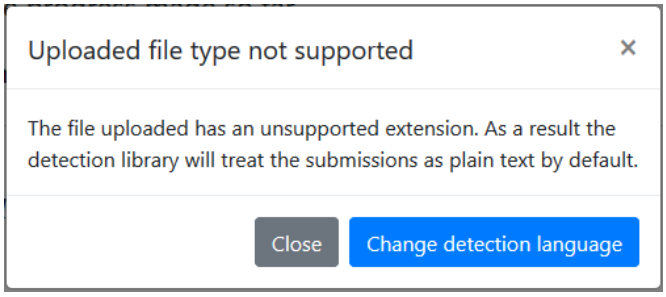


Figure 3.8: Unsupported file type dialogue

The similarity detection threshold option, allows the similarity detection to be more lenient or strict depending on the user's preference. After experimenting with different thresholds, the value of 80% similarity is used by default, as similarities above that value strongly suggest a connection between the segments and the submissions. This feature and the reason for its addition is further discussed in Chapter 4.

3.3.2 Upload Page

The upload page follows the welcome page and is where the user is instructed to upload the suspect file and the folder that includes all of the submissions.

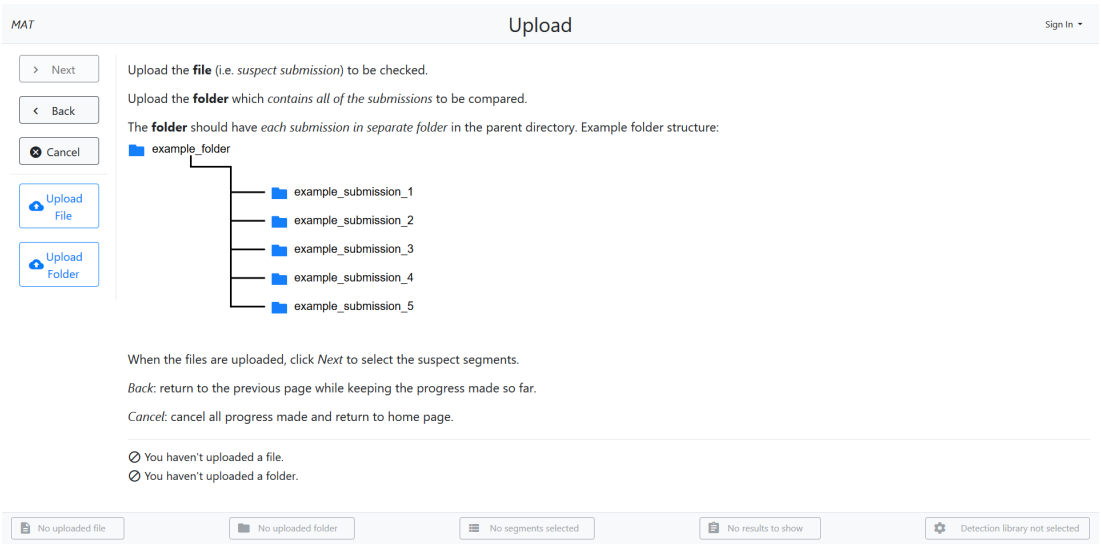


Figure 3.9: Upload Page

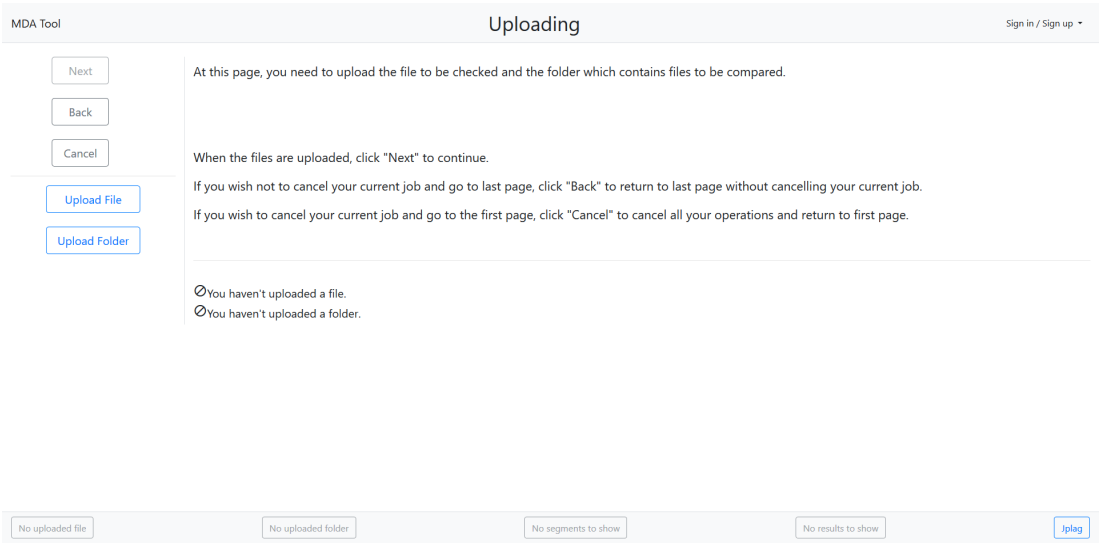


Figure 3.10: Previous system's Upload Page

First, this iteration tackled what is considered a usability issue to new users. The inherited system included unclear instructions which made it difficult for an unfamiliar user to upload the folder containing all of the submissions in the correct form. Because the implementation assumes a given structure, uploading a folder in a different structure would result in incorrect results, without any way to verify their correctness. The needed structure requires to upload the parent folder that has each submission in a different folder. However, it is normal to assume that some users would upload a single folder that has all the submission files inside. When this case happens, the system considers the upload as having zero submissions because it counts the number of sub-folders and there are none. Hence, the expectation estimate calculation would be wrong and the user wouldn't know why (Section 2.2). This issue was addressed with two methods, having improved instructions and providing context of how the upload is being interpreted.

The instructions now are more useful, as they specify what has to be uploaded explicitly, leaving less room for doubt. An example image of the submission folder structure was also added, as it is easier to convey the folder structure visually to the user (Figure 3.9).

Additional context is now provided after the upload of the files. Users are able to determine easily if they uploaded the correct submissions as the number of submissions is indicated next to the upload status after uploading a folder. In the previous system, it was noticed that it was impossible to know if an upload of the wrong structure was made (Section 4.1). Now the user can immediately tell if the number shown is different than expected, thus avoiding miscalculated results from accidentally wrong submissions. The number of submissions is retrieved client-side by a response to the `POST` request, which was already being used to upload the folder.

Another feature that was added in this iteration, is the detection of whether the submitted suspect file is included in the uploaded folder. When both the file and folder are uploaded, the system compares all the files in the folder with the suspect file. This

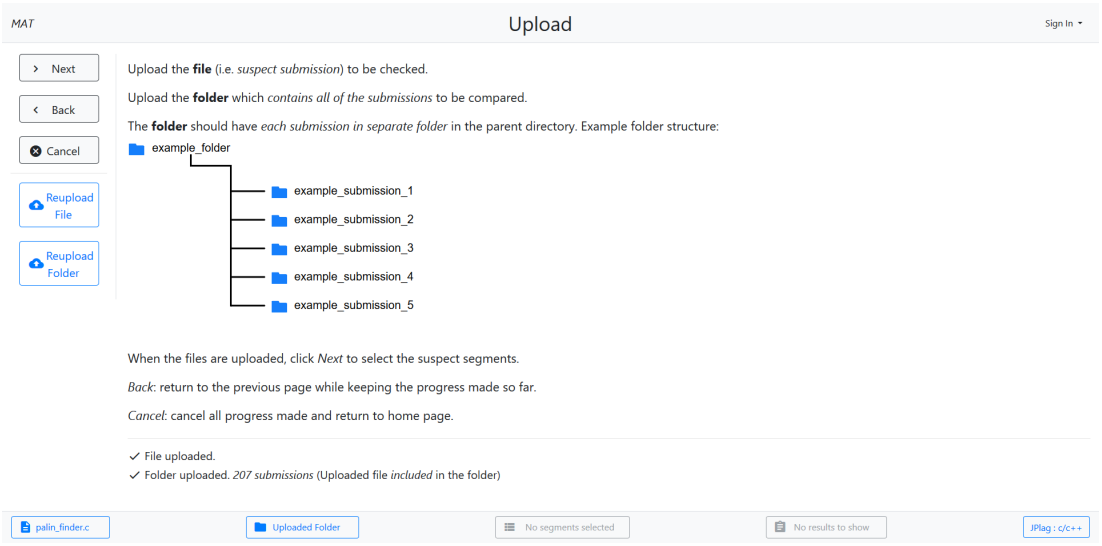


Figure 3.11: Upload Page after uploading the suspect file and the submissions folder

information is displayed next to the number of submissions. In the previous system, this feature was missing and always assumed the uploaded suspect file was included in the uploaded folder without instructing the user. This information is critical, as the resulting calculation of the expectation estimate would be incorrect (Section 2.2). With this additional detail, the correct adjustments are made when calculating the results (Section 3.3.5). The client retrieves this information by a `GET` request to the newly configured `upload/checkIncluded/` URL on the Django server. The response is either *Yes*, *No* or *NA* (Not Available) in the case that either the suspect file or submissions folder is not uploaded yet.

Another improvement over the previous iteration was the update of the bottom context bar immediately after making an upload, i.e. enable the uploaded file and folder buttons. The previous iteration used a Django's [12] *Template Context Processor* to give the context each time a page is accessed using the *Template* engine. The Template engine provides a convenient way to generate HTML dynamically adding the content server-side before sending it to the client. However using this approach, the uploaded file and uploaded folder buttons on the bottom context bar would not be updated until a new page is loaded. This proved to be unproductive as immediate feedback [54] is essential and users should be able to examine and verify the files uploaded immediately. The direct update of the context bar is implemented by introducing `GET` method `upload/updateContext/` in the back-end RESTful API. When a request is made, the response returns the updated context using the same method that is used in the Template Context Processor.

Finally, the uploaded folder button located in the bottom context bar, points to the newly created *Uploaded Folder Page* (Figure 3.12). In the previous system, the uploaded folder button revealed a popover that included the list of files in the submissions folder. However, the popover proved to be hard to use when a large number of submissions is made because of its smaller size, thus a smaller number of submissions can be viewed without scrolling. Further, after feedback from the project's supervisor

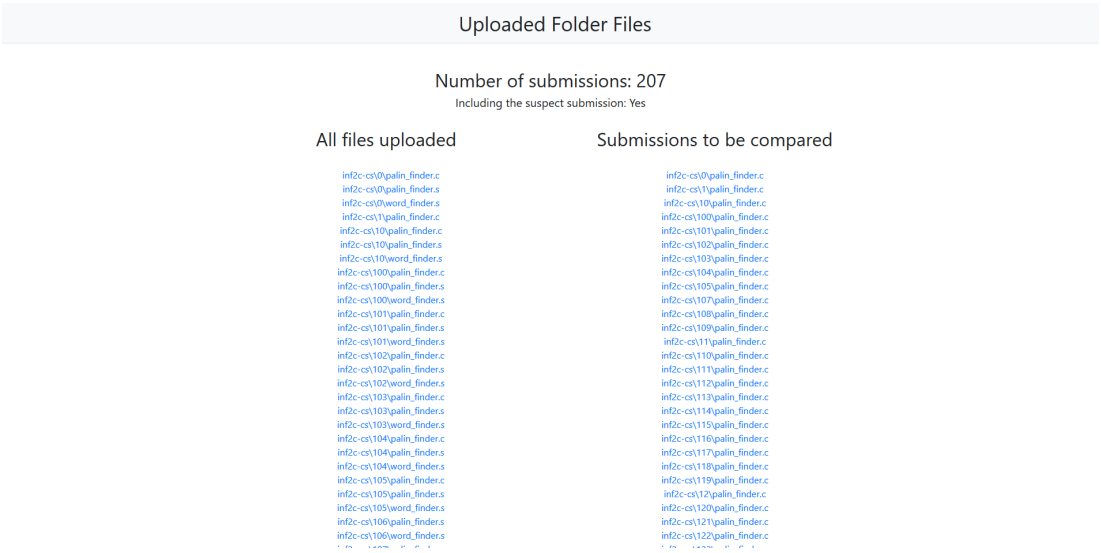


Figure 3.12: Uploaded Folder Page

Kyriakos Kalorkoti which is the School's Academic Misconduct Officer and the main user of this project, it was considered more consistent [54] to have the list open in a different browser tab, similar to the suspect file button action. Also, he found it useful to list the files that are being considered in the detection separately, i.e. displaying only the source code files written in the same language as the suspect file.

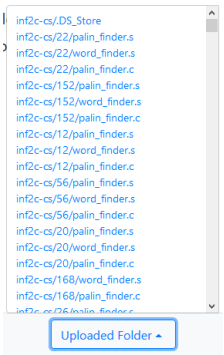


Figure 3.13: Previous system's Uploaded Folder popover

3.3.3 Segment Selection Page

The segment selection page comes after the upload page and is where the suspect segments for misconduct are chosen. The suspected submission's code is shown in the left part of the content section and on the right are the selected segments (Figures 3.14 & 3.16) to be used for the expectation estimate (Section 2.2). The user selects the lines of code for a section and presses the middle "Add selected segment" button, to add that selection as a segment and highlight it with a different colour. The user can add some additional lines to a segment by selecting them and pressing the "Append selection" button. Segments can be removed using the "Delete segment" button

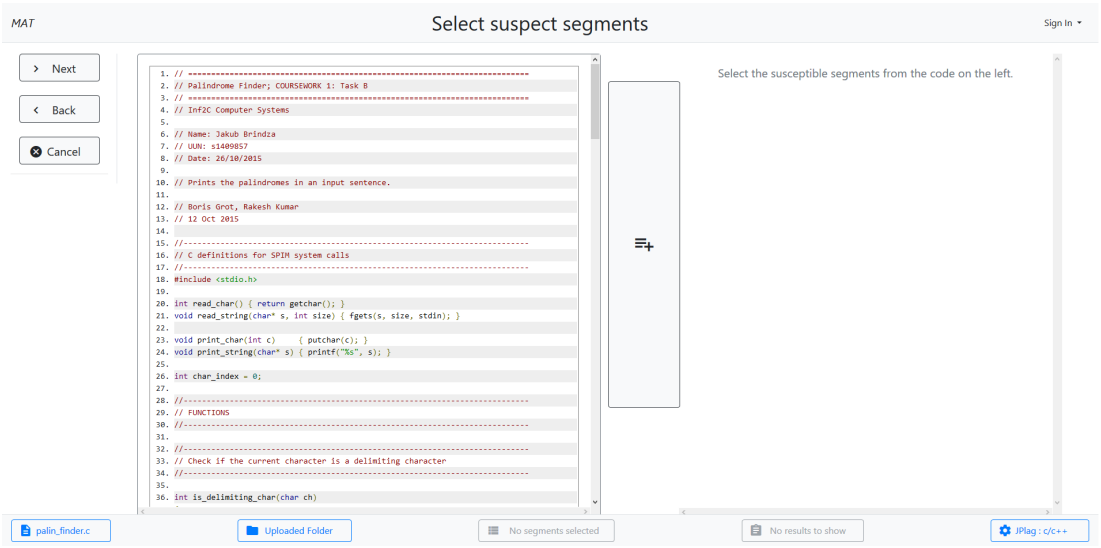


Figure 3.14: Segment Selection Page

and they can be excluded from the estimation calculation by unticking the "Included:" checkbox.

First, the *save button* from the inherited system was removed as it was deemed ambiguous. A user could press *Next* and not select save, but the segments would be saved. However, pressing *Back* or reloading the page would discard the unsaved segments. Now, each action is automatically saved, thus leaving no room for doubt.

The use of buttons which have no text is unclear to some users [36]. In this iteration, all of those buttons use tooltips (information shown when the cursor hovers over them) to explain their functionality.

An important usability change in the Segment Selection Page was making the action buttons on the left navigation pane and the "Add selected segment" button stays *fixed* in the screen while scrolling down. In the inherited system, when scrolling down, the buttons would advance upwards. Hidden from the scrolled down view, made it difficult to add a new segment or continue "Next" to the Results Page (Figure 3.18). Thus, the solution was to make the left navigation pane and the "Add selected segment" button *sticky*, i.e. stay immobile while scrolling (Figure 3.19). Note that in smaller windows that don't fit both the suspect submission's code and the segments side by side, the page is restructured vertically and the "Add selected segment" button's position is changed accordingly (Section 3.3.1, Figure 3.6).

For the implementation, the left side panel and the "Add selected segment" button use the *CSS* property `position:fixed`. In order to have dynamic restructuring for a smaller window size, two separate buttons were made and appear according to the page size using Bootstrap's responsive tiers (Section 3.3.1). For bigger screens the button includes the classes `.d-none`, `.d-xl-block` which make it visible only on window widths greater or equal to 1200*px*), and `position-fixed` for the fixed position while scrolling. For smaller screens the button included the classes `btn-block d-xl-none` instead. Class `btn-block` makes the button block level, i.e. spanning the full width

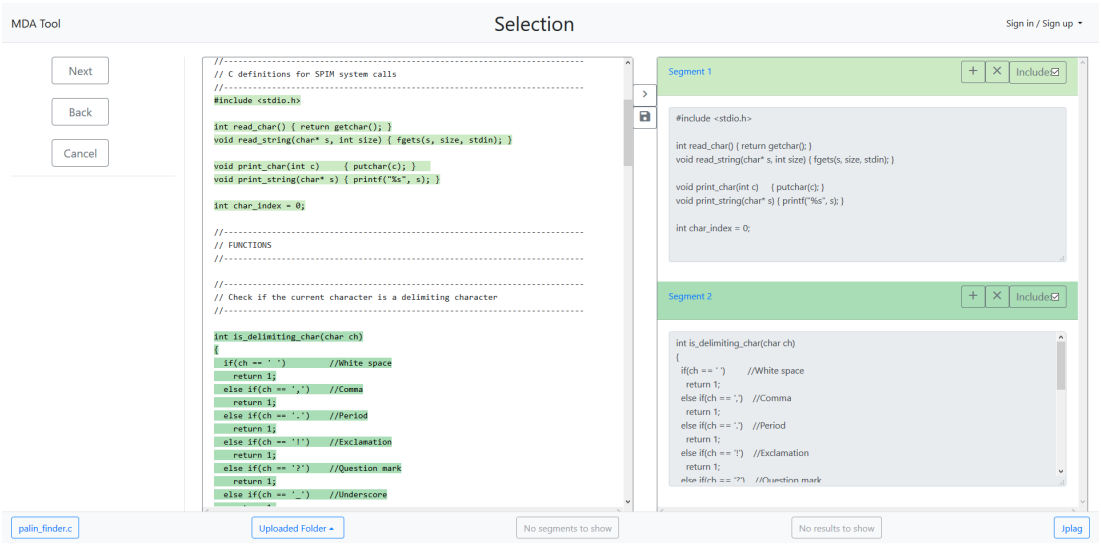


Figure 3.15: Previous system's Segment Selection Page with some added segments

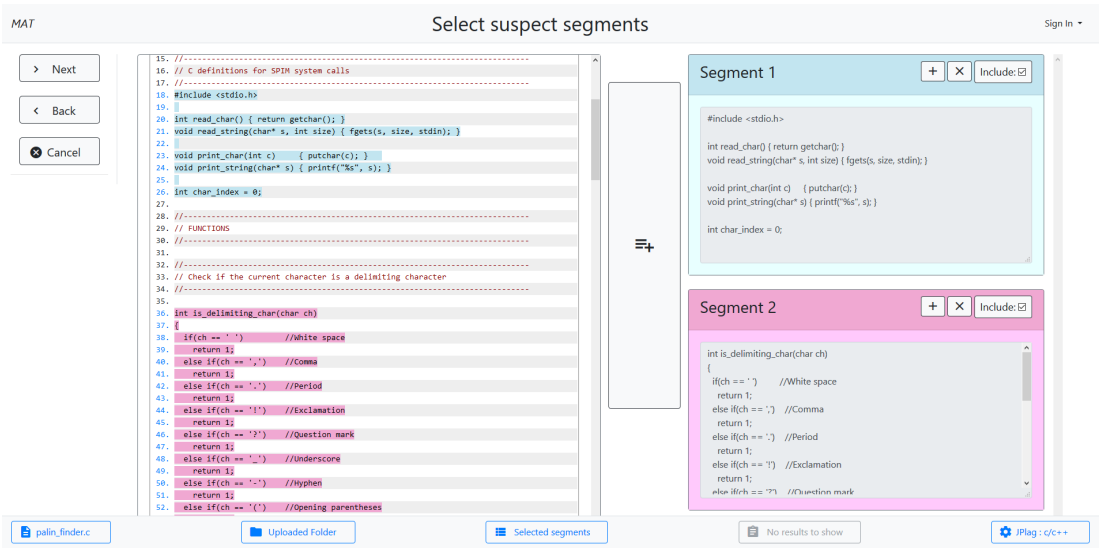


Figure 3.16: Selection Page after adding segments

of the content section. Class `d-xl-none` is used to display it for screen sizes less than $1200px$.

The "Add selected segment" button was also made bigger indicating it's importance. When pressed, it adds the selected code as a segment and highlights the particular code section. Making no selection or selecting less than five lines of code (excluding comments), emphasises the code section with a red outline for five seconds and an instructional tooltip is shown to help guide the user to select a segment that is valid (Figure 3.20). In the previous system, it was unclear how to select and add a segment, however now, because of the bigger "Add selected segment" button and the tooltips, it is much easier to understand how to add a segment.

When the button is pressed, the selected text is retrieved and passed to the method `checkSelectedText(text)`, which replies with whether there are less than five lines.

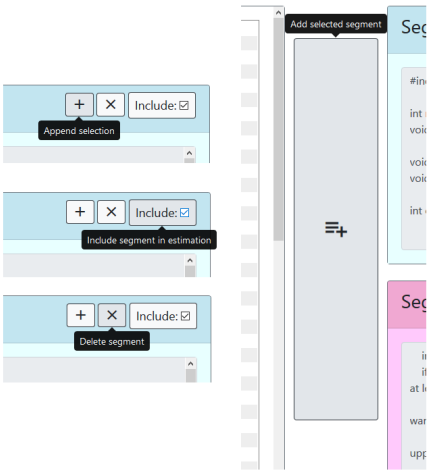


Figure 3.17: Tooltip buttons in Segment Selection Page

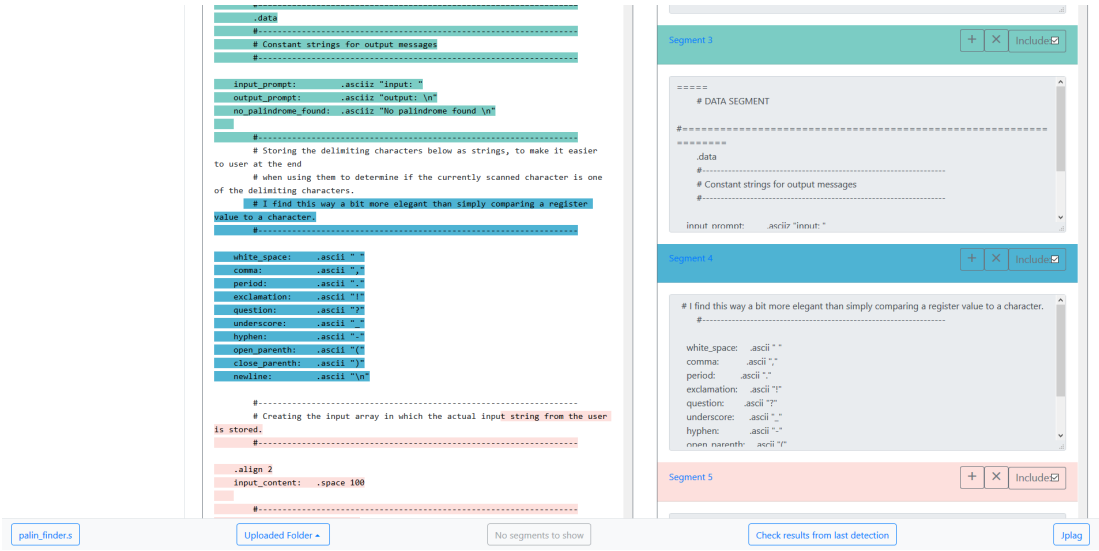


Figure 3.18: Action buttons hidden when scrolling down in the inherited system

It counts the number of code lines, by splitting the text into lines on the line-break character, trimming the leading white-space and checking if the first character is alpha and not a comment symbol because the similarity detection packages ignore commented out lines (Section 4.2). When less than five lines of code are selected, the function `displayTooltip()` is called, otherwise the segment is added, highlighting it and drawing the segment section (Figure 3.16). The displayTooltip() function adds Bootstrap's `border` and `border-danger` classes to the code section element creating the red emphasis border and removes them after five seconds. Also, it enables and shows the instruction tooltip. The choice of requiring five or more lines of code is because the JPlag [49] similarity detection package is too sensitive if configured to detect such small sections. Further discussion given in Chapter 4.

Another improvement was the formatting of the code section with *syntax colouring*. It was accomplished using *JavaScript Code Prettifier* [27], which is an embedded library script that makes source-code snippets in HTML prettier. Because the use of

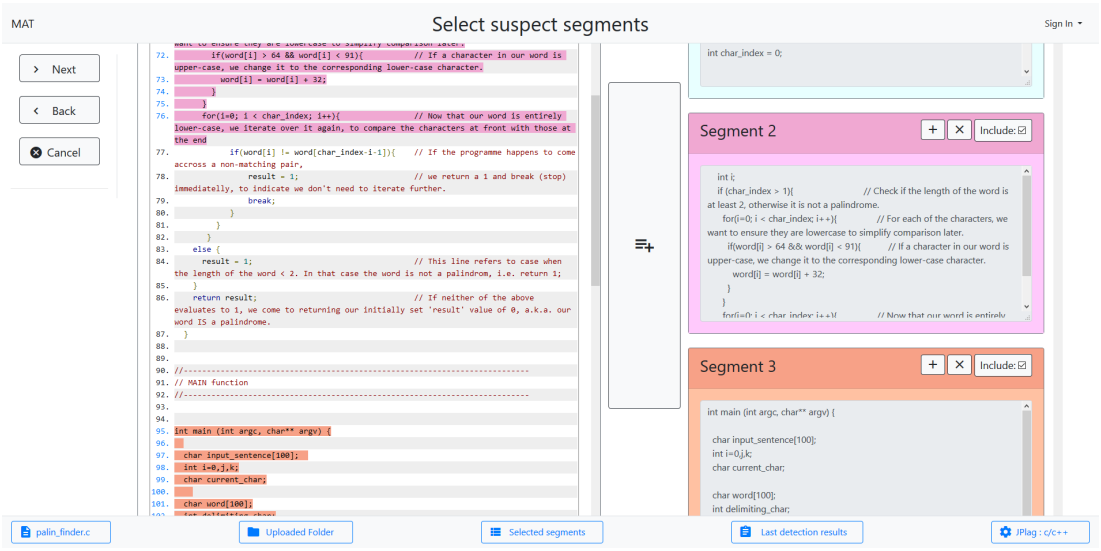


Figure 3.19: “Fixed” buttons in the new system

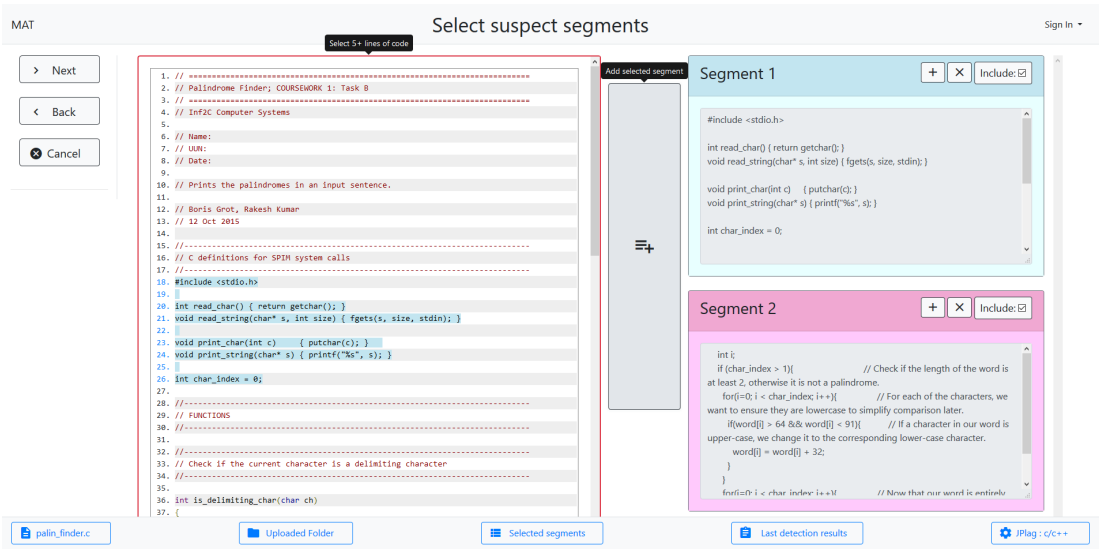


Figure 3.20: Emphasised code section after attempting to add empty selection

this library interfered with the existing highlighting mechanism, some changes were made to retain segment highlighting in the code section. The inherited system used `window.getSelection()` and `window.getSelection().getRangeAt(0)` for getting the selected code and its range. It then highlighted the segment by creating a new anchor element (i.e. `<a>`), used to surround the selected range. This element has a background colour inferred from the segment number and a hyperlink to its segment section. The code section with highlighted segments was saved by saving the HTML code in a file back-end. When the *Code Prettifier* “prettifies” the code section it wraps each line with a *list item* (i.e. `<li>`) and each differently coloured token with an *inline* element (i.e. `<span>`) (Listing 3.1). Each wrapper element uses a class to assign the font and background colour (Figure 3.21).

```
<li class="L0">
    <span class="kwd">void</span>
    <span class="pln"> read_string</span>
    <span class="pun">(</span>
    <span class="kwd">char</span>
    <span class="pun">*</span>
    <span class="pln"> s</span>
    <span class="pun">,</span>
    <span class="pln"> </span>
    <span class="typ">int</span>
    <span class="pln"> size</span>
    <span class="pun">)</span>
    <span class="pln"> </span>
    <span class="pun">{</span>
    <span class="pln"> fgets</span>
    <span class="pun">(</span>
    <span class="pln">s</span>
    <span class="pun">,</span>
    <span class="pln"> size</span>
    <span class="pun">,</span>
    <span class="pln"> stdin</span>
    <span class="pun">);</span>
    <span class="pln"> </span>
    <span class="pun">}</span></li>
<li class="L1">
    <span class="pln"> </span>
</li>
```

Listing 3.1: Sample “prettified” code in HTML form

```
21. void read_string(char* s, int size) { fgets(s, size, stdin); }
22.
```

Figure 3.21: Sample “prettified” code from Listing 3.1 in rendered form

The added wrappers broke the inherited highlighting method because it couldn’t simply wrap an anchor element in the new structure using `window.getSelection()` straight away, as it has to wrap all the *list item* elements. The `getSelectionRange()` method was created to return the selected text from the beginning of the line to the end of the last line, the corresponding range, the start node and the end node objects. The start and end nodes are found by using the JQuery’s [4] `closest("li")` method to the anchor (start) and focus (end) nodes given by `window.getSelection()`.

The anchor node indicates where the selection started. When the selection is made starting from the end to the start (i.e. bottom-up), the start node is going to be after the end node. To remove ambiguity when the nodes are used in the following methods, the start node has to be the first node in the selected code section. Thus, the start and end nodes are compared using JQuery’s index function on the list elements. If the start node has a bigger index then the two are swapped.

Once this information is gathered, it is passed on the method `highlightCode(selectedTextRange, startNode, endNode, segmentNumber)`. It creates a new anchor element that wraps the list elements (lines), linking them to the corresponding segment section. Also, it highlights each node in that range by using the method `highlightNode(node, colour)`, which finds every *inline* element (`<span>`) in the node (`<li>`) and changes its CSS background property to the given colour. The font is turned to black, because the different highlight colours make some syntax colours difficult to read, e.g. dark green code with a light green background.

Additionally, the segments section has been restructured with each segment having a different Bootstrap [45] *card* which is a flexible and extensible content container. In the header of each *card*, the segment's title links to the corresponding highlighted code section on the left. Also in the header, the "Append selection", "Delete segment" buttons and "Included:" checkbox. The colours of the segments have been changed from the inherited system to more contrasting with each-other making it easier to uniquely identify them. The palette of colours is colour-blind friendly, as approximately 1 in 12 men and 1 in 200 women are colourblind [11]. They have been tested using *Color Oracle* [34] that simulates Deuteranopia (Figure 3.22), Protanopia and Tritanopia forms of colour blindness.

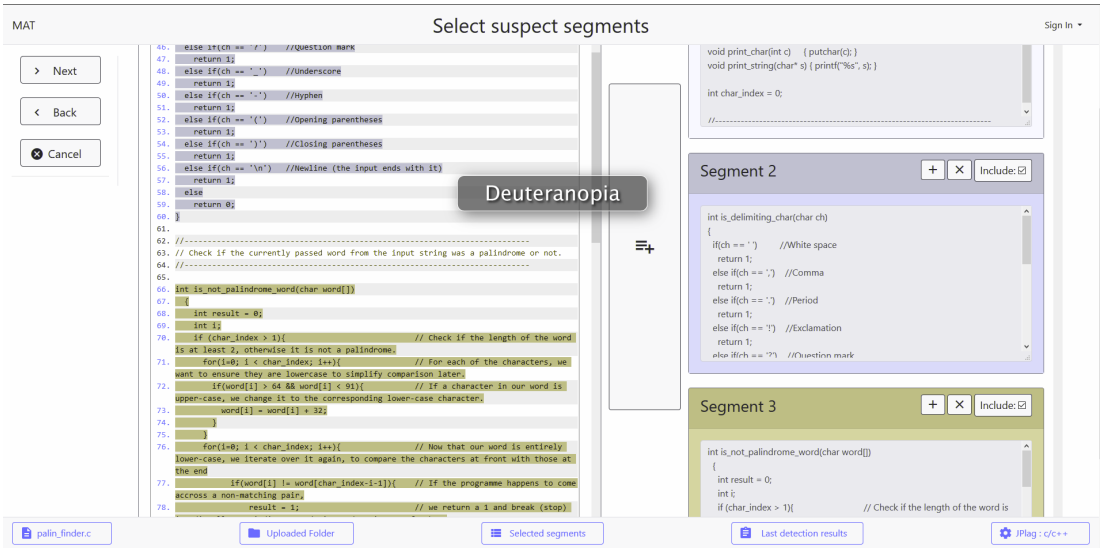


Figure 3.22: Deuteranopia simulated system

Finally, the inherited system had some software bugs (Chapter 4) that are now repaired. One of these allowed the user to use the "Next" button with no segments selected, which resulted in an unhandled error. The problem was solved by implementing the `areSegmentsEmpty()` method, that checks whether any segments exist. Method `updateNextButtonStatus()` uses `areSegmentsEmpty()` to enable or disable the "Next" button accordingly. It is invoked by every action that modifies the segment sections e.g. deleting a segment. Hence, the "Next" button is now always disabled if no segments selected are present.

3.3.4 Waiting Page

The waiting page is shown while the code similarity detection package (Section 2.3.2) processes the given segments with the submission folder files. When the process finishes, the user is directed to the results page (Section 3.3.5).

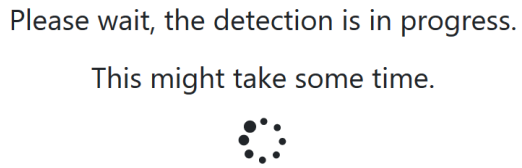


Figure 3.23: Waiting Page

Although the front-end design of the waiting page remained the same, the processing of the given suspect segments has changed in this iteration. After testing the previous system (Chapter 4), it was determined that the default *Minimum Match Length* (sensitivity) used with JPlag [49] was too imprecise for small segments, which are a crucial part of the Misconduct Assessment Tool. The previous iteration used the default sensitivity value 12, which after testing it was determined that sections with less than 12 instructions were not matched. For example, having 4 sensitivity, only sections of a segment matching minimum 4 instructions with other submissions are considered. The use of the *Minimum Match Length* parameter from JPlag was determined through testing (Section 4.2), as no documentation of JPlag is available during this iteration of the system (September 2018 - March 2019).

The current system divides the segments into two groups. Segments that have between 5 and 11 instructions are in the *small group*, while the ones with 12 or more instructions are in the *normal group*. Segments smaller than 5 are not allowed as noted in the Section 3.3.3, as the testing determined that the sensitivity would have to be too low, matching almost all other submissions (Section 4.1). The division of the segments is done using their line count and the lists of the two groups are saved into the session's folder as a Python's pickle file to be loaded when the results from JPlag (HTML form) are to be interpreted each time the results page is accessed. JPlag is executed twice if both small and normal section is present. Segments in the *small group* are processed with a sensitivity of 3, while segments in the *normal group* use sensitivity 6.

3.3.5 Results Page

The results page is where the user can inspect the results of the investigation. After the similarity detection package finishes processing the suspect segments and its results are interpreted, the expectation estimate (Section 2.2) is calculated and the detailed results are shown. Additionally, in this iteration the user can generate a PDF report of the results, a feature further discussed in Section 3.4.

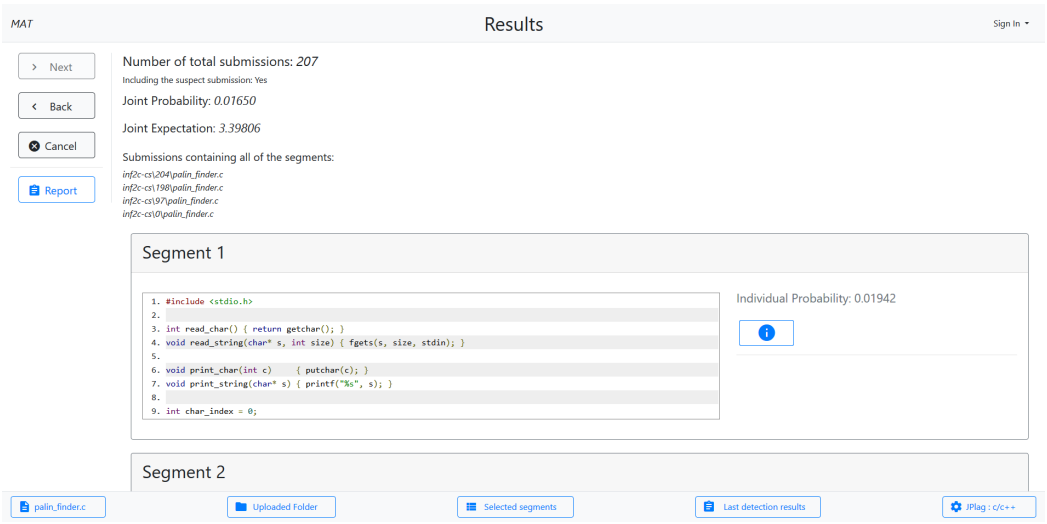


Figure 3.24: Results Page

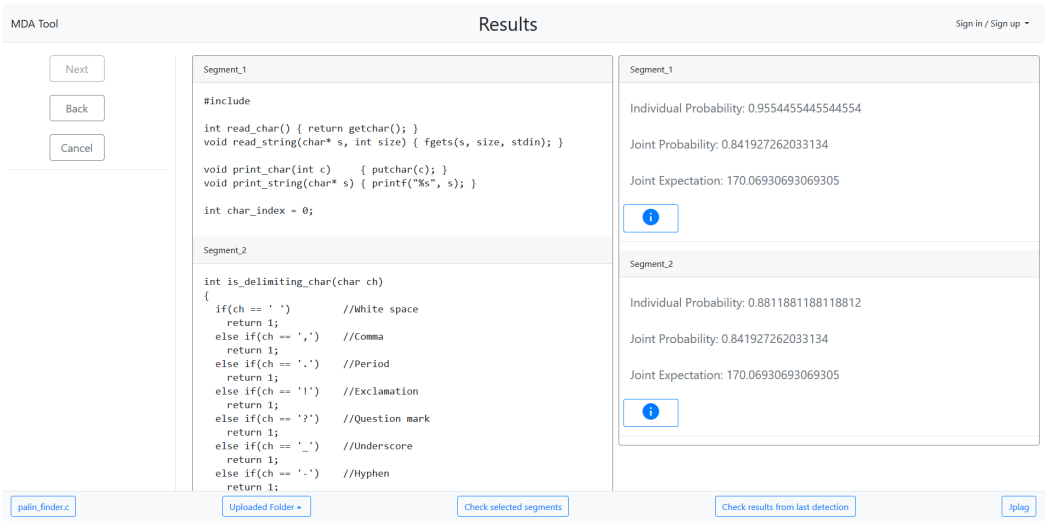


Figure 3.25: Previous system's Results Page

Firstly, the current iteration restructured the results shown on the page in a more meaningful way. The previous iteration repeated the *joint probability* and *joint expectation* values along with the *individual probability* for all segments. However, the *joint probability* and *joint expectation* can be considered as the general results as they are deduced from all the segments. Hence, they were moved to the top of the page and given more emphasis due to their importance.

Additionally, more context is also given at the top of the page now, helping the user in the analysis of the results. The number of submissions and whether the suspect file is included in the calculation can also be used to verify that the results were calculated using a correct submission because the user can easily see a wrong number of submissions with an accidentally wrong uploaded submissions folder. The inclusion of the suspect submission is now taken into account when calculating the expectation estimate, unlike the previous iteration which always assumed that the suspect file is included and used $N-1$ in all the calculations (Section 2.2).

The submissions that are *detected to have all the investigation segments*, are now found and displayed. This extra information was added to further assist the investigation by providing a more in-depth analysis of the results. In the previous iteration without the extra context, an expectation estimate greater than one, usually suggested that the suspect submission is expected by chance. However, with the added context, if more than one submissions share those same segments, further investigation is recommended especially if the expectation estimate doesn't match the number of those submissions.

Those submissions are found when iterating over all the segments when calculating the expectation estimate (Listing 3.2). The `filesWithAllSegments` *set* is initialised with the files that have the first segment. For each subsequent segment, the files included in the *set* are filtered to only include the ones that also appear for this segment (working similarly to a logical *AND*). If a segment has no matching files then the set is emptied.

```
// For each segment
for (i = 1; i <= maxSegment; i++) {
    ...
    // Check whether the current segment has any matching submission files
    if (suspectFilesKeys.length !== 0) {
        ...
        // Determine the files has the same segments as the suspect
        if(!filesWithAllSegments){
            // First call, initialise filesWithAllSegments
            filesWithAllSegments = new Set(suspectFilesKeys);
        }else{
            // Only include the files that also appear for this segment
            let currentSegmentFiles = new Set(suspectFilesKeys);
            filesWithAllSegments = new Set(
                [...filesWithAllSegments].filter(x =>
                    currentSegmentFiles.has(x))
            );
        }
    }else{
        // No matching files for this segment, thus no file in general
            will have all the segments
        filesWithAllSegments = new Set();
    }
}
```

Listing 3.2: How submissions that appear in all segments are found in `results.js`

As noted in the Waiting Page section 3.3.4, this iteration introduced the division between small and normal segments of the suspected code. The session's small and normal files list is loaded from the previously saved Python's pickle file (Section 3.3.4). The previous process of interpreting the resulting HTML files from JPlag was refactored, the existing code was extracted as the method `process_result_file( results_path, search_files, path_folder)`. This new function was created because JPlag is executed twice if both small and normal sections are present and its results are saved into a "small" and "normal" directory. Hence, this function is called with an adjusted `results_path` and `path_folder` depending the directory being considered. The parameter `search_files` includes the list of segments that are in the set being considered.

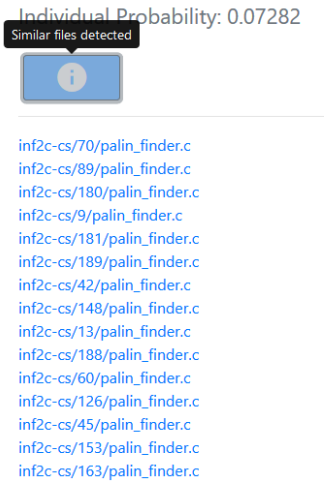


Figure 3.26: A segment's similar files detected

Finally, the results page was restructured for consistency [54] similar to the restructuring of the segment selection page (Section 3.3.3). The code segments are formatted with *syntax colouring* using *JavaScript Code Prettifier* [27]. More importantly, each segment is now using it's own Bootstrap [45] *card* similar to the segment selection page. In the previous iteration, the segments and their results information were separated making it hard to match as the user had to find the corresponding sections manually. However, in this iteration each segment card contains the code, the individual probability of that segment (Section 2.2) and the files that have been found to contain a similar segment (Figure 3.26). Also, the button that reveals those files is now annotated with a tooltip, helping users and keeping a coherence for buttons with no text.

3.4 Report Generation

The new report generation feature makes sharing the results of the investigation easy and convenient. Instead of printing the web-page with the results, or taking screenshots of it, the users of the system are now able to export the results of the investigation. The format of the report chosen was PDF because it is standardised [13], platform

independent and widely used. This is important because the results are intended to be shared with the School's Misconduct Officer and the rest of the responsible parties.

When designing the report it was important that it was *consistent* with the web representation, a core principle in interface design [54]. Thus, the report contains the information presented in the results page (Section 3.3.5) using a similar format.

People reading the report, like the student being investigated and external parties, might not understand its purpose and further clarification is thus included. The produced report includes a brief introduction stating the purpose of the tool and how the results can be interpreted. The introduction includes a crucial point, that the results are not conclusive, but only used for assisting the decision making progress.

Apart from the *Introduction* segment (Figure 3.27), the rest of the report is dynamically created using the results of the detection and the details given in the report generation form (Figure 3.28). Optional information, like the notes section, can be used to include more context to the report because each investigation case is different. The user can choose segments in the report to be brief, which means that the code of each segment will not be included in the report, otherwise the detailed option includes them.

The only required field is the student's matriculation number used for unique identification. If the matriculation number is not included (Figure 3.28b) then the generation is cancelled and the required field is highlighted red to draw attention to the user as well as a note is shown to explain that it has to be filled. The validation is implemented using Bootstrap's [45] custom `valid` and `valid` CSS classes that are applied to all of the input fields used in the form. These are triggered using JavaScript by adding a '`submit`' *Event Listener* to the form (indicated by the `needs-validation` class in the HTML document). When the event is triggered, the validity is checked and the form is validated by reassigning the form's class to `was-validated`, changing the CSS styling of each input element according to a given `valid` / `invalid` CSS class. The default form submission method was disabled as it refreshed the page unnecessarily, clearing the form after a submission. By preventing the page refresh, the modification of the form's fields is faster, permitting easy reversal of actions [54].

When the user submits the form pressing the `Generate` button, the report is created and opened in a new tab to be viewed by the user. The report generation is implemented using the JavaScript library jsPDF [5]. Thus, it is generated client-side, relieving the server of extra processing. The inherited system calculated the *expectation estimate* (Section 2.2) in JavaScript too, making it more convenient to process those values directly, instead of passing them to Django (which is the server-side back-end processor).

The jsPDF is an open-source library to generate PDFs. It uses a *xy-axis* positioning system to determine where text is to be written. Because of this system, a number of supplementary functions were created in this development cycle to aid with the creation of the report document in a format similar to any document preparation system like LateX [6] and word processor Microsoft Word [7].

First, the page height `page_height` and width `page_width` are retrieved from jsPDF so that the positioning of the text will be calculated relative to those measurements. The margins of the page are given by `v_margin` and `h_margin`. Using the margin the

Academic Miscoduct Report

Student: John Doe (s123456)
Course: Inf2b - Algorithms, Data Structures and Learning

Introduction

This report is produced by the Misconduct Assessment Tool produced within the School of Informatics, University of Edinburgh.

The aim of this tool is to test student submissions that are not obvious cases, but suspected of misconduct due to multiple small segments being similar to other submissions. With a large number of submissions, it is fairly likely that some segments will be similar by chance. This tool is used to help in such situations, by estimating the expected number of submissions with that exact combination of segments. It is built to assist the decision making process, *it does not replace it.*

For each suspect segment, the individual probability of that segment is calculated by taking the number of submissions that have a similar segment over the number of all the submissions. Segments that are present in a lot of submissions have a higher individual probability. The joint probability is then calculated by assuming that each segment is independent and taking the product of all the segments' individual probabilities. Finally, to calculate the estimated number of submissions we take the joint probability and multiply it with the number of all the submissions. The resulting number should be the number of students anticipated to have the exact combination of the suspected segments. If the expected number is lower than one, a case of misconduct is suggested. Please note that the results are not conclusive.

General Evaluation

Number of total submissions: 208
Including the suspect submission: Yes

Student's questionable segments

Joint Probability: 0.00050
Joint Expectation: 0.10385

A joint expectation < 1 suggests no student is anticipated to have this combination of segments.

Submissions containing all of the segments:

inf2c-cs\0\palin_finder.c
inf2c-cs\0copy\palin_finder.c

Segment 1

Individual probability: 0.02415
Number of similar submissions: 5

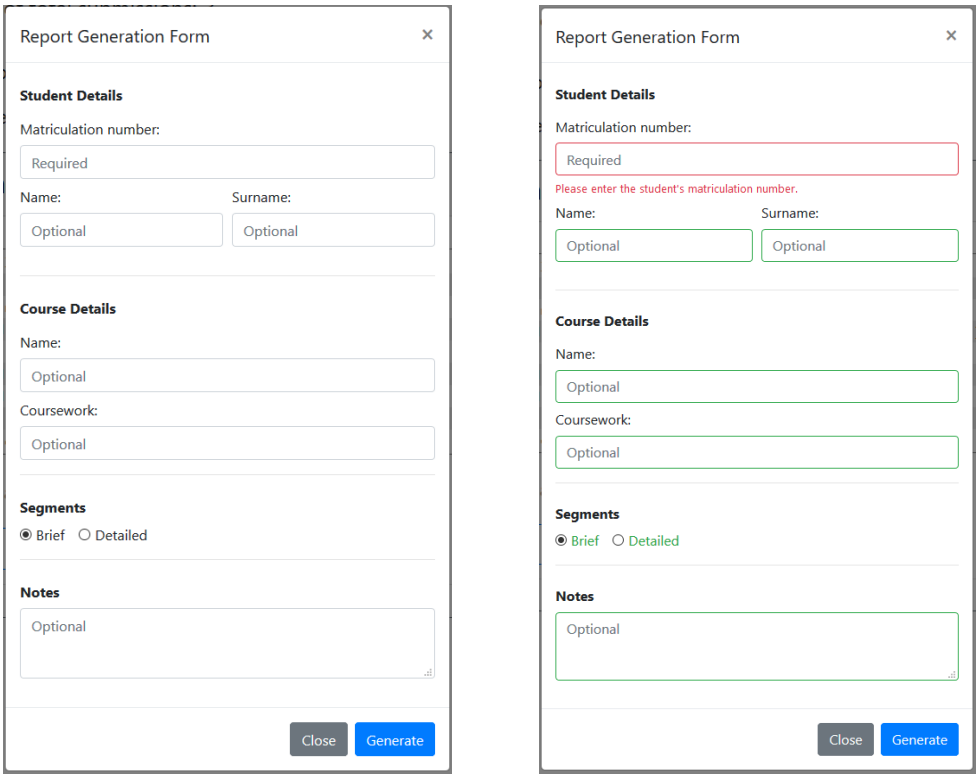
```
#include <stdio.h>

int read_char() { return getchar(); }
void read_string(char* s, int size) { fgets(s, size, stdin); }

void print_char(int c)    { putchar(c); }
void print_string(char* s) { printf("%s", s); }

int char_index = 0;
```

Figure 3.27: The first page of a generated report



(a) Empty form (b) Validated Form

Figure 3.28: Report generation form

`line_width` is determined by `page_width - (h_margin * 2)`.

The positioning of each printed text is given by the `x` and `y` variables that are initialised with the `h_margin` and `v_margin` respectively. Some of the most important functions implemented were:

- The `print_long_text(long_text)` function that splits the text into lines that are printed inside the margins using `print_text(line)`.
- The `print_text(text)` prints a single line and manages the line spacing by adding the `font_size/2` to `y`. The function also automatically adds the page numbering footer and changes the page when `y > page_height - v_margin`.

These, along with other supplementary functions like `separate_sections()` as well as `add_footer()`, make it easier to add content dynamically without having to worry about finding the correct coordinates for each action. Furthermore, it allows future additions to be easily implemented because of its modularity.

3.5 Server Deployment

The Misconduct Assessment Tool being a web system, it was important to be deployed for the main user and project supervisor Kyriakos Kalorkoti the School's current Aca-

demic Misconduct Officer. The deployment of the system meant that users would be able to access the system without having to deploy their own local instance and get any updates of the system remotely and immediately. Before, the deployment the user had to get a local copy of the system's files and manually run the server to use the system. During this iteration, a virtual server has been requested and provided by the School of Informatics to run the system as a service in the School's network.

Initially, the system was to be deployed as for production, using an *Apache HTTP Server* [1] with the *mod_wsgi* [23] module to host Python web applications. However, the virtual server access provided didn't have root access or any documentation which made the typical setup of the server much more difficult. Hence, after pursuing different attempts to do the Apache deployment, considering the timeline and priorities for this iteration, it was chosen to deploy the server using *Django*'s [12] development web server. This takes into account that the web application in the current development iteration is only being regularly accessed by the developer and the main user Kyriakos Kalorkoti, thus the security and performance increase is not as important at this stage. Nonetheless, at later stages the end goal is that the system is available widely, at least to all the School of Informatics staff (Section 5.1).

First of all, before starting the deployment of the web server, all the necessary dependencies had to be gathered in the virtual server. The dependencies associated with Python were exported as an `environment.yml` file from the development environment using Anaconda [14] and loaded into a new *Anaconda virtual environment* on the virtual server (virtual environments keep these dependencies in "sandboxes" which can be easily switched or shared).

Next, the *JPlag* similarity detection package used, is written in *Java* [16], so it needed Java Runtime Environment to run. As a result, the required Java Runtime Environment was retrieved and scripts that set and unset the system environment variables associated with the binaries of Java were made. The *Shell* scripts (Listings 3.3 & 3.4) are located in the `etc/conda/activate.d/` and `etc/conda/deactivate.d/` directories of the environment and are executed when activating / deactivating the *Anaconda virtual environment*.

```
#!/bin/sh
export JAVA_HOME=/disk/data/s1509375/java/jre1.8.0_191/bin/java
export PATH=$PATH:/disk/data/s1509375/java/jre1.8.0_191/bin
export SECRET_KEY=<some secret key>
```

Listing 3.3: Activation script

```
#!/bin/sh
unset JAVA_HOME
unset PATH
unset SECRET_KEY
```

Listing 3.4: Deactivation script

Django uses a secret key to provide cryptographic signing. Because data passes through an untrusted medium, cryptographic signing is critical as it is used to detect any tampering now that the User Accounts have been introduced (Section 3.6). The secret key is being passed to the deployment using the `SECRET_KEY` system environment variable, set in the same script as the Java environment variables (Listings 3.3 & 3.4). The secret key is set to a unique, unpredictable value, as it is crucial to be kept secret.

Finally, the `settings.py` file of the Django project had to be modified to load the secret key from the system environment variable and the allowed hosts parameter had to include the new domain that the system is deployed (Listing 3.5).

```
SECRET_KEY = os.environ['SECRET_KEY']
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', 's1509375.inf.ed.ac.uk']
```

Listing 3.5: Modified parameters in settings.py

The deployed service can be accessed from the School of Informatics internal network at the address: http://s1509375.inf.ed.ac.uk:8000/. The previous iteration has also been deployed for reference at: http://s1509375.inf.ed.ac.uk:8080/. Note that deployment is in debugging mode for easier troubleshooting. Once the system is ready to be used by the School of Informatics staff, the debugging mode should be turned off.

3.6 User Accounts

Given that the Misconduct Assessment Tool is aimed to be used by multiple users not only the School's Academic Misconduct Officer, *user accounts* was one of the future goals suggested by the previous developer [60]. As the previous iteration was found to provide unreliable results (Chapter 4) and had usability issues (Section 3.3), the priorities were shifted and the user accounts feature was considered a secondary goal. As a result, the base of the user authentication system has been implemented, but the user session storage feature is left to be implemented in a future iteration (Chapter 5.1).

For the implementation of user accounts, Django's [12] user authentication system was used. It handles both authentication (verifies who a user claims to be) and authorisation (determines what an authenticated user is allowed to do). In this iteration, only the authentication part is being exercised.

The table used for storing the user accounts is stored in an *SQLite* [33] database. *SQLite* is a relational database management system (DBMS) that is open-source, self-contained and file-based. It was chosen for this iteration over other DBMS like the more popular *MySQL* [8, 10]. Unlike, *MySQL*, *SQLite* is a "serverless" database, which means that it simplifies the setup processes and it is faster to prototype and test on. Because the system is still in early development, it would be unnecessary to setup a DBMS host server, as the concurrency, security and speed advantages associated with it are not relevant yet.

The *Users* table (Figure 3.29) is used to store details about each user of the system.

Users	
PK	**username**
	password
	email
	first_name
	last_name

Figure 3.29: Users Table Structure

For this iteration only username and password are being populated. However, the other fields are included for a future iteration (Section 5.1) where the user accounts feature is going to be expanded. The primary key (required and unique) field *username* may contain up to 150 *alphanumeric*, _, @, +, . and − characters. The *password* field is also required and stores the *hash* of, and metadata (salt, algorithm, iterations) about the password. Passwords should never be stored raw in case of a system breach, thus the raw password goes through the one-way hashing algorithm PBKDF2 [56] with SHA256 [43] and a random salt to prevent the use of rainbow tables in a potential attack to reveal the hashed passwords.

A registered user can login using the sign in drop-down menu that is revealed using the sign in button on the top title bar (Figure 3.30). When logged in, the sign in button instead shows the user's username and when clicked the account menu is shown with the option to logout (Figure 3.31). The account menu was used instead of just changing the sign in button to a logout button because the account menu could add other account related options like account settings in the future and it also stays consistent [54] with its previous use of opening a drop-down menu.

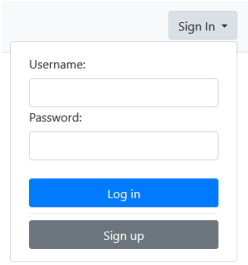


Figure 3.30: Sign in drop-down menu

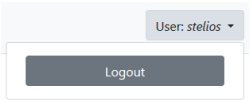


Figure 3.31: Account drop-down menu

The implementation of the sign in and the account menus utilises the *Django template language* (DTL) to generate the HTML document dynamically. Using DTL eases the design as it defines control logic and accesses variables directly from HTML. When serving a page to a user, the choice of which of the two buttons and drop-down menus to show, are controlled by the `is_authenticated` attribute of the user variable (Listing 3.6).

```
<div class="btn-group" id="login-form-group">

    <button type="button" class="btn btn-light dropdown-toggle"
        data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
        {% if user.is_authenticated %}
            User: <i>{{ user.get_username }}</i>
        {% else %}
            Sign In
        {% endif %}
    </button>

    <div id="login-dp" class="dropdown-menu pull-right
        dropdown-menu-right px-4 py-3">
        {% if user.is_authenticated %}
            <a class="btn btn-secondary btn-block" href="{% url 'logout'
                %}?next={{ views.index }}">Logout</a>
        {% else %}
            <form class="" method="post" action="{% url 'login' %}">
                {% csrf_token %}
                <label for="id_username">Username:</label>
                <input class="form-control" id="id_username"
                    type="text" name="username" maxlength="30"/>
                <label for="id_password">Password:</label>
                <input class="form-control" type="password"
                    name="password" id="id_password"/>
                <br/>
                <input type="submit" class="btn btn-primary btn-block"
                    value="Log in"/>
                <input type="hidden" name="next" value="{{
                    request.get_full_path }}"/>
            </form>
            <div class="dropdown-divider"></div>
            <a class="btn btn-secondary btn-block" href="{% url 'signup'
                %}?next={{ request.path }}">Sign up</a>
        {% endif %}
    </div>
</div>
```

Listing 3.6: Sign in / Account menu

When users are authenticated (logged in), then their username is shown using the method `get_username` of the `user` variable, and the logout button is added to the dropdown menu. The logout button accesses the logout page, which then uses the `next` *GET* query parameter and redirects to the index of the system defined in `urls.py`, that in the current iteration is the Welcome Page (Figure 3.2).

When users are not authenticated, then the login form is shown. The login form is sent using an "unsafe" POST method and for this reason, it includes a *CSRF* token which is handled by Django's [12] CSRF middleware. This token protects users against cross-

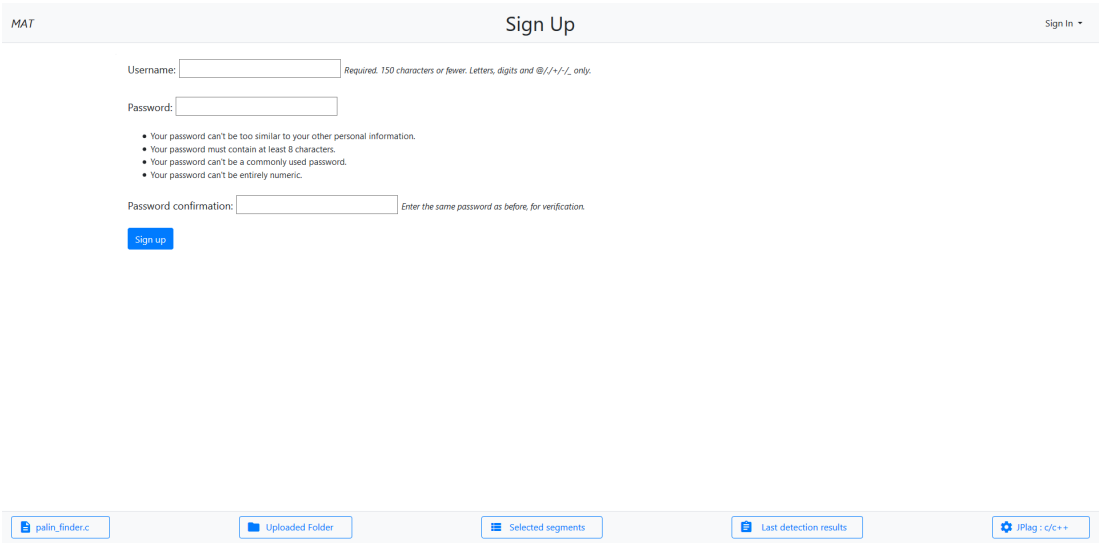


Figure 3.32: Sign Up page

site request forgery (CSRF) attacks, where a malicious site can cause a visitor to make an unauthorised request, which then the server thinks is authorised because it contains the users' cookies. Under the login form is the sign up button, that sends the users to the sign up page (Figure 3.32). It includes a `next` *GET* query parameter with the current page path (`request.path`), so that when the users sign up, they get back to the current page. The sign up page is constructed similarly using DTL. It has instructions helping the user choose a fairly secure password and basic password validation rules for rejecting too simple ones (less than 8 characters or only composed from digits).

3.7 Logging Mechanism

As the system is now deployed, a logging mechanism was deemed necessary. An event log is now created to record events taking place in the execution of the system. These events can then be used to understand the activity of the system and diagnose problems. For example, an uncaught exception causes an *Internal Server Error* to one of the user's actions. This exception will then be logged with enough information to further investigate its cause. If no log is kept, then there would be no way to investigate problems encountered by the users.

The logging setup uses Django's [12] and consequently Python's builtin logging module. The logger is added and configured in `settings.py` (Listing 3.7). Apart from showing the results in the console using a `logging.StreamHandler`, the log is saved to file, using a `logging.FileHandler`.

```
LOGGING = {
    ...
    'formatters': {
        'timestamped': {
```

```
            'format': "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s]
                %(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        }
    },
    'handlers': {
        ...
        'file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': './mdp.log',
            'formatter': 'timestamped'
        }
    },
    'loggers': {
        'django': {
            'handlers': ['console', 'console_debug_false', 'file'],
            'level': 'INFO',
        },
        ...
    }
}
```

Listing 3.7: Logging to file configuration in settings.py

There are five different log severity levels, going from `DEBUG` to `CRITICAL`. The file handler was setup to log all events with `INFO` severity and above (ignores `DEBUG` messages). `INFO` messages were inserted in several parts of the system (e.g. `logger.info("Interpreting results from JPlag")`) to give some context of activity to potential `ERROR` events.

Finally, the log events are rendered into text using the `timestamped` formatter (Listing 3.7), which adds a timestamp before each event message to help pin down user reported problems (Listing 3.8).

```
[20/Mar/2019 13:29:59] ERROR [django.request:228] FileNotFoundError:
    [Errno 2] No such file or directory:
    'misconduct_detection_app\\uploads\\127.0.0.1\\segments\\Segment_'
[20/Mar/2019 13:35:05] ERROR [django.request:228] Internal Server Error:
    /select/selectCode/
Traceback (most recent call last):
  File "C:\Users\steli\Anaconda3\envs\mat\lib\site-packages\django\core\
      handlers\exception.py", line 34, in inner
    response = get_response(request)
  File "C:\Users\steli\Anaconda3\envs\mat\lib\site-packages\django\core\
      handlers\base.py", line 126, in _get_response
    response = self.process_exception_by_middleware(e, request)
  File "C:\Users\steli\Anaconda3\envs\mat\lib\site-packages\django\core\
      handlers\base.py", line 124, in _get_response
```

```
    response = wrapped_callback(request, *callback_args,
        **callback_kwargs)
  File "C:\Users\steli\Desktop\Coding\Miscoduct-Detection-Project\
      misconduct_detection_app\views.py", line 370, in select_code
    os.makedirs(get_segments_path(request))
  File "C:\Users\steli\Anaconda3\envs\mat\lib\os.py", line 221, in
      makedirs
    mkdir(name, mode)
FileExistsError: [WinError 183] Cannot create a file when that file
    already exists:
    'misconduct_detection_app\\uploads\\127.0.0.1\\segments'
```

Listing 3.8: Example logged events

Chapter 4

Testing

This chapter discusses the testing of the inherited system, some of the problems found and details on how they were fixed. Additionally, an investigation of the similarity detection package was done and is also described in the chapter.

4.1 Inherited System

Given that this is the second iteration of the Misconduct Assessment Tool [60], the system had to be tested and verified that it worked as intended and is sufficiently usable. The timeline was limited, thus the testing was primarily black-box system testing. This approach evaluated the system's compliance with the specified requirements using different use-cases. The testing was done by both the project supervisor and School's Academic Misconduct Officer Kyriakos Kalorkoti and the software developer. The black-box [26] method of testing used, meant that the functionality of the system was tested without any regard of the internal implementation (source code). Only when the behaviour was unexpected, the source code was then examined.

In addition, the system has been evaluated in term of usability and ease of use and understanding. The issues found are discussed in Section 3.3.

For the testing environment setup, two different datasets approaches were used. The first approach was a dataset comprised of real submissions from an older year course-work and provided in confidentiality by the project supervisor. This dataset was used to test the system in a real situation where there are a lot of submissions with variable similarities between them. The second approach was an artificially made dataset that had some completely different code files with some of them having multiple exact copies to simulate obvious plagiarism or collusion and test that the calculation of the expectation estimate is correct (Section 2.2).

First, regular use of uploading a file selecting a few segments and analysing the results was tested.

Uploading a submissions folder with a different structure than the sub-folder separated

format explained in Section 3.3.2, results in wrong calculations and the user is unaware of it. In Figure 4.1 the uploaded folder included the submissions' files in the folder's root directory (no separate sub-folders). From examining the code, the system detected zero submissions and thus no submissions were compared to the suspect file. The users have no way of realising that these results are wrong. As discussed in Section 3.3.2, the issue was addressed with two methods. Having improved instructions that clearly state how the uploaded folder should be structured as well as providing feedback on how the upload is being interpreted by displaying the number of detected submissions in the Upload Page and not allowing users to proceed to segment selection if zero submissions have been detected (disabled next button).

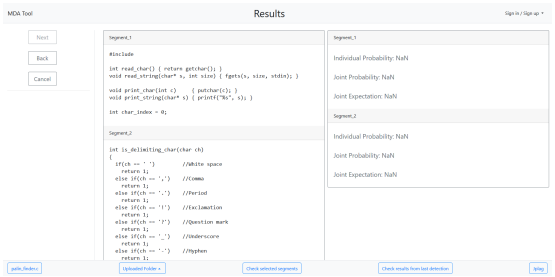


Figure 4.1: Results after an unknowingly wrong upload

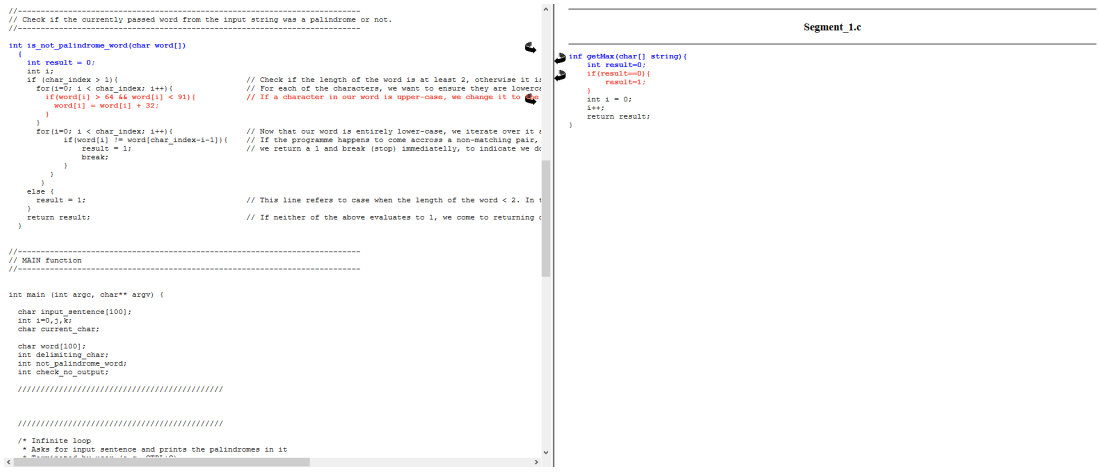
Next, it was tested whether excluding the suspect submission from the uploaded folder would adjust the calculation of the expectation estimate when a correctly structured submissions' folder is uploaded. For testing this case the uploaded folder included one submission that was completely different from the suspect file, and two submissions that were almost identical (included some extra lines), thus would be detected as similar. The resulting individual probability (Section 2.2) for a segment was $p = \frac{2}{2}$ instead of the correct $p = \frac{2}{3}$. This meant that the denominator used $N - 1$ instead of N which implied that every upload was assuming that the suspect file is included. This iteration automatically detects if the suspect file is included and lets the user know as soon as the folder is uploaded and the calculations are adjusted appropriately (Section 3.3.2).

Then when testing different segment sizes, a dataset that included exact copies of a submission was used so that the similarity is guaranteed. After experimentation, it was determined that quite small segments were not being considered in the similarity detection and always reported an individual probability of zero. This was considered a serious issue because the main use case of this system involves selecting a number of small segments.

At first, the results from the JPlag [49] similarity detection package (Section 2.3.2) were inspected to ensure that the problem was not involved with their interpretation from the Django [12] back-end server. After verifying that JPlag did not detect the similarity, the parameters that JPlag was being executed with were investigated. JPlag has a parameter named Minimum Match Length (sensitivity), considered very relevant to the problem faced, that wasn't specified in execution and thus the inherited system used the default value of 12. Some testing had to be done to determine how does this parameter work, because of JPlag's limited documentation (Section 4.2). It was

determined that sensitivity value 3 (Figure 4.2b) would be best appropriate for small segments that are under 12 instructions. Otherwise, sensitivity value 6 would be best for the bigger segments. Using a lower sensitivity value for these two cases results in extremely sensitive detection given their context with single instructions matching in any part of the document (Figure 4.2a). The implementation of this solution is discussed in Section 3.3.4.

(a) Minimum Match Length (sensitivity) = 1

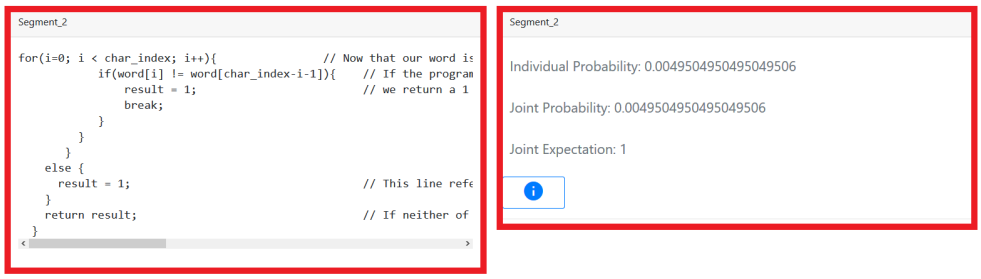


(b) Minimum Match Length (sensitivity) = 3

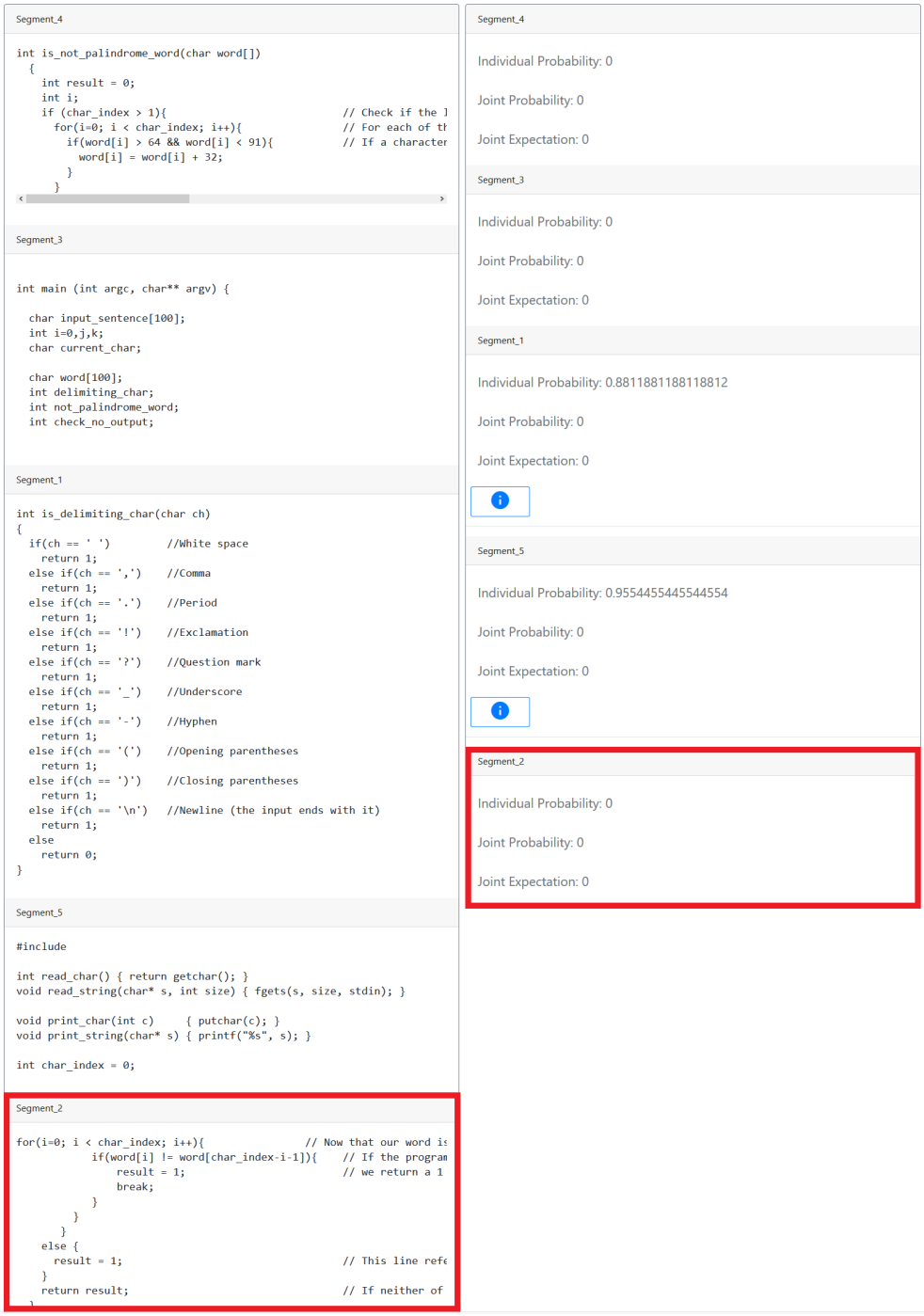
Figure 4.2: JPlag results with low sensitivity

Furthermore, while testing with a number of real submissions, it was noticed that adding more segments in the investigation altered the results for some segments, thus being unreliable. More specifically when a segment was tested alone in the investigation, it would show matching submissions and have an individual probability greater than zero (Section 2.2). When tested with a number of other segments selected, then that *same* segment would detect no matching submissions and would have an individual probability of zero (Figure 4.3).

Similar to before the first thing that was inspected were the JPlag results files. From there it was clear that with a larger number of segments, the number of matches were capped to 20. That meant that if more than 20 matches are detected, only the top 20



(a) Segment 2 investigated alone



(b) Segment 2 investigated with other segments

Figure 4.3: Segment detection inconsistency

will be included (in order of descending similarity percentage). This is a result of using the default *Matches* parameter when running JPlag. Because the results have to be consistently correct throughout use, a *threshold* of the similarity percentage is now given, and as a consequence removing the limit of the number of matches displayed. This iteration uses the *threshold* parameter with a default value of reasonable 80% similarity, which can be changed by the users depending on how strict the similarity between segments should be (Section 3.3.1).

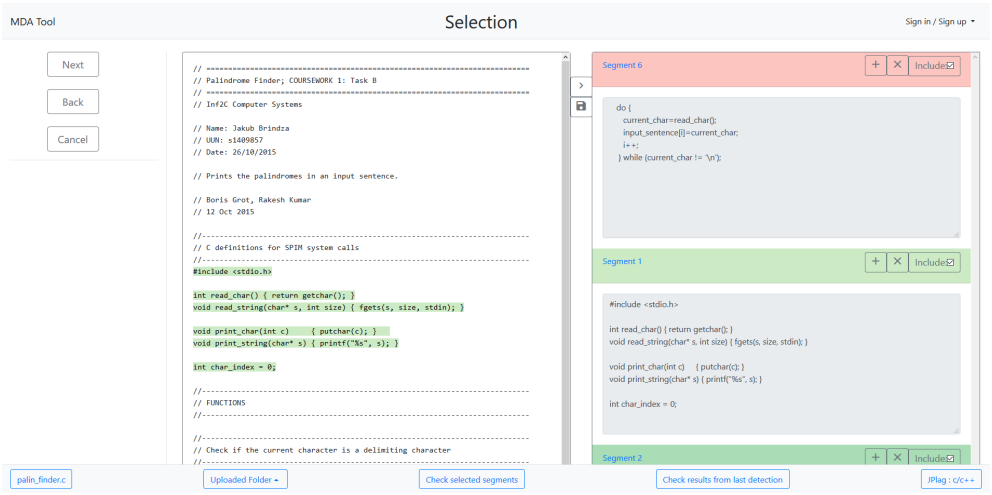


Figure 4.4: Segments ordered randomly in the segment selection page

While testing with a number of segments it was noticed that those segments were not displayed in an ascending order in the results page (Figure 4.3b). The same problem was also found in the segment selection page. When the segments are added they appear in order. However, if the user revisits the page (refresh or coming back from another page) the segments appear unordered (Figure 4.4). After examining the source code, it was clear that the reason of this random order was that the segments were read using the `forEach` loop over the segments' filenames from the `segmentFiles` dictionary's keys. In order to resolve this imperfection, the maximum segment index had to be found (Listing 4.1). Then to display each segment, a `for(i=1;i<=maxSegment;i++)` loop used to access each segment in ascending order. Because, there's the option to delete segments resulting in some missing segment indices, a condition is added that checks if the segment with the index used exists before trying to display it.

```
var maxSegment = 0;
segmentFilesKeys.forEach(segmentKey => {
    // Extract the segment index e.g. 1 from Segment_1.c
    let currentSegment =
        segmentKey.substring("Segment_".length).split('.').slice(0,
        -1).join('.');
    let currentNumber = parseInt(currentSegment);
    if (maxSegment < currentNumber) maxSegment = currentNumber;
});
```

Listing 4.1: Search for the maximum segment index

Moreover, once some segments have been selected, every time a user went back to the segment selection page, the suspect file code would add an extra empty line between each line. An initial inspection of the code did not indicate any problems with the saving and loading the suspect code. Once some segments have been selected and highlighted, the system saves the HTML form of the suspect code in order to retain the highlighted parts and links to the segments. This is where the problem was. When inspecting the produced HTML with visible control characters, it was clear that the newline characters *carriage return* plus *line feed* (`CR+LF`) were added unnecessarily in addition to the original *carriage return* ones found in the suspect code (`CR`) (Figure 4.5). This bug would be undiscovered if this development iteration didn't use a Microsoft Windows operating system, as this *newline* character is one of the differences between Unix, Mac and Microsoft Windows based operating systems. The solution to this problem was simply changing the newline parameter when opening the HTML file in write mode using Python. The default `None` value translated the `CR` character into the OS's native `CR+LF` while also leaving `CR` which is unwanted. The `''` (empty string) option used now does not do any translation and fixes the problem.

```
//--------------------------------------------------------------------------CR
CRLF
// C definitions for SPIM system callsCR
CRLF
//--------------------------------------------------------------------------CR
CRLF
<a style="color: black; background: #ccebc5" href="#highLightedSegmentHeader0" id="highLightedSegment1">#include &lt;stdio.h&gt;CR
CRLF
CR
CRLF
int read_char() { return getchar(); }CR
CRLF
void read_string(char* s, int size) { fgets(s, size, stdin); }CR
CRLF
CR
CRLF
void print_char(int c)     { putchar(c); }   CR
CRLF
void print_string(char* s) { printf("%s", s); }CR
CRLF
CR
CRLF
int char_index = 0;</a>CR
CRLF
```

Figure 4.5: Suspect code in HTML form with visible control characters

Unfortunately, a problem found to be appearing only when using a Safari browser wasn't fixed in this iteration. The project supervisor Kyriakos Kalorkoti tested the system with the Safari browser and when he attempts to upload some files in the Upload page (Section 3.3.2), the uploads are not registered by the system. Because the developer had access only to a Microsoft Windows and the School's DICE Linux machines, it wasn't possible to use the Safari browser and find the exact cause of this problem. As a result, this problem is left to be solved in a future iteration where the developer has access to a Mac machine. Otherwise, the system was tested by the project supervisor on the same machine as Safari with both Firefox and Chrome and it was found to be working correctly.

Finally, some other problems found while testing were:

- Reuploading a different suspect file after already selecting segments would not change to the new file and would keep the previous suspect file and selected segments. Fixed by removing the previously uploaded file and selected segments when uploading a new suspect file.

- In the case of a suspect file having multiple dot (.) characters in its filename, it would read the wrong file extension resulting in a wrong programming language detection. It was resolved by finding the *last* instead of the *first* dot character in the filename.
- The user would be allowed to press next while some uploading wouldn't be finished resulting in a faulty state where either the suspect file or the submissions folder was not uploaded. The problem was solved by enabling the next button only after the asynchronous JavaScript and XML (AJAX) request that uploads the file/folder finishes successfully.
- The user would be able to choose next without choosing any segments in the Segments Selection page (Section 3.3.3). This was fixed by updating the status (enabled/disabled) of the next button whether a segment is added or deleted.

It should be noted that after each introduced change (Chapter 3) the system was tested to ensure that things remained working and no unexpected side-effects had been established. An example would be when the results between small and normal sized segments were separated, the links to each result analysis (Figure 3.26) were broken and had to be fixed by pointing to the new locations.

4.2 JPlag Similarity detection package

As discussed in the previous section, the JPlag [49] similarity detection package (Section 2.3.2) has some parameters that had to be adjusted for the use of this system. Unfortunately, there is no available documentation on JPlag, as when trying to access the URL of the official client documentation[1] it leads to an error page (Figure 4.6).

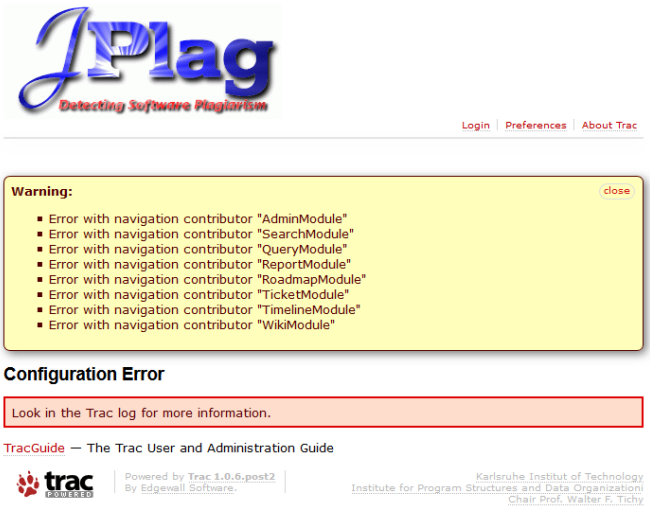


Figure 4.6: JPlag documentation page error (accessed March 2019)

[1] https://svn.ipd.kit.edu/trac/jplag/wiki/ATUJPlag/Client

README.md

Download and Run JPlag

Download a released version - all releases are single-JAR releases.

Type `java -jar jplag-yourVersion.jar` in a console to see the command line options. The options as of 2017/09/17 are:

```
JPlag (Version 2.11.9-SNAPSHOT), Copyright (c) 2004-2017 KIT - IPD Tichy, Guido Malpohl, and others.
Usage: JPlag [ options ] [<root-dir>] [-c file1 file2 ...]
 <root-dir>        The root-directory that contains all submissions

options are:
 -v[qlpd]        (Verbose)
                 q: (Quiet) no output
                 l: (Long) detailed output
                 p: print all (p)arser messages
                 d: print (d)etails about each submission
 -d              (Debug) parser. Non-parsable files will be stored.
 -S <dir>        Look in directories <root-dir>/*/<dir> for programs.
                 (default: <root-dir>/*)
 -s              (Subdirs) Look at files in subdirs too (default: deactivated)

 -p <suffixes>   <suffixes> is a comma-separated list of all filename suffixes
                 that are included. ("-p ?" for defaults)

 -o <file>       (Output) The Parserlog will be saved to <file>
 -x <file>       (eXclude) All files named in <file> will be ignored
 -t <n>          (Token) Tune the sensitivity of the comparison. A smaller
                 <n> increases the sensitivity.
 -m <n>          (Matches) Number of matches that will be saved (default:20)
 -m <p>%         All matches with more than <p>% similarity will be saved.
 -r <dir>        (Result) Name of directory in which the web pages will be
                 stored (default: result)
 -bc <dir>       Name of the directory which contains the basecode (common framework)
 -c [files]      Compare a list of files.
 -l <language>   (Language) Supported Languages:
                 java17 (default), java15, java15dm, java12, java11, python3, c/c++, c#-1.2, char, text, sch
```

Figure 4.7: JPlag's repository `README` information

The only source of information left for JPlag was its GitHub[2] repository's `README` file. Shown in Figure 4.7 is all the information regarding the use of each parameter.

Sensitivity parameter

The sensitivity parameter is considered mainly for the *small* segment similarity detection. From Figure 4.7, the sensitivity parameter is `-t <n>` and is described as: "(Token) Tune the sensitivity of the comparison. A smaller `<n>` increases the sensitivity.". From this, it is unclear how the sensitivity works and how should it be adjusted for the Misconduct Assessment Tool.

An analysis of how the sensitivity parameter works was made through testing different source code files. Previous experimentation with JPlag suggested that the sensitivity was linked to the size of each segment getting matched. The files tested had an increasing number of instructions starting from just a single instruction to sixteen different instructions, each one building up from the previous keeping the same starting instructions (Figure 4.8).

Then, JPlag was executed multiple times with all the different sensitivity values from

[2]https://github.com/jplag/jplag

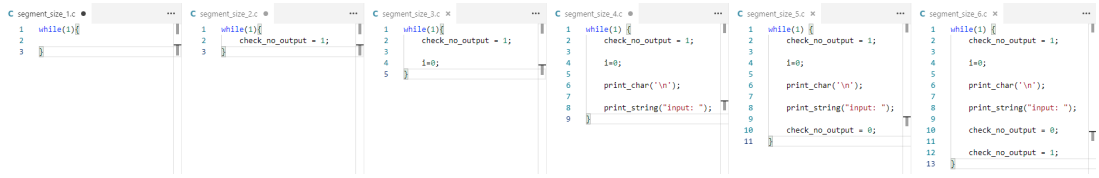


Figure 4.8: Files with increasing number of instructions used to test sensitivity

1 to 16. From analysing the results it was clear that the sensitivity indeed directly connected to the number of instructions being considered when detecting similar segments. With a sensitivity value 1, all of the files matched other files, while with sensitivity value 2 all the files except from the file with a single instruction were getting matched. This pattern continued until the final sensitivity value 16 where no files were matched because there's only one file containing 16 instructions.

(a) Sensitivity = 1

(b) Sensitivity = 2

(c) Sensitivity = 15

(d) Sensitivity = 16

Figure 4.9: Results from different sensitivity values from JPlag

Also to ensure that it was instructions and not lines of code that are being considered, the file that had 8 instructions was copied and an extra instruction in the same line as another instruction was added. Examining the results from the run with sensitiv-

ity 8 and 9, it is suggested that JPlag considers instructions and not lines. The extra instruction in the file made it appear in the sensitivity 9 results, while the original 8 instruction file did not appear. The file that had 10 instructions was also copied and additional comments were added to observe if there's a different treatment from the original. From the results, it can be concluded that comments are not taken into account as it appeared in the same exact results as the original file.

Chapter 5

Conclusion

The Misconduct Assessment Tool is a unique system that uses the ability of existing similarity detection packages to detect similarities between code [49] to test the less obvious cases of misconduct. Situations where a student submission has multiple small segments being similar to other submissions are where the Misconduct Assessment Tool comes into use by estimating the expected number of submissions with that exact combination of segments.

As explained in Chapter 2, this project builds upon the system initially developed by Y. Xie [60]. In this iteration of the project, new features were added while also creating a more robust and reliable system. Putting my work into perspective:

- Results can now be generated as a PDF report.
- The system has been deployed for use inside the School of Informatics internal network.
- User accounts have now been introduced.
- Improved reliability and consistency of the system results by dividing segments into small and normal sized as well as by using the similarity threshold parameter.
- Revised the underlying structure of the interface to support different screen sizes.
- Enhanced the user experience by redesigning the interface. More context and helpful information were added and usability issues were fixed.
- Improved the understanding of how the JPlag similarity detection package interprets several parameters through extensive testing.
- Tested the inherited system and provided solutions to a number of bugs.

All tasks originally planned have been completed and any suggestions along the development from the project supervisor have been implemented.

Working closely with the project supervisor gave me real-world insight about the software development life cycle. Evolving requirements as well as continuous analysis, de-

sign, and implementation were all part of the development of the project. Even when challenges were faced, it was still exciting and intriguing to see the system develop and improve step by step. All in all, this project has been an invaluable experience and helped me develop tremendously as a software engineer.

5.1 Future Work

Given the time constraints involved with the project, the current version of the Misconduct Assessment Tool includes the most relevant and useful features to the supervisor. However, as with most software development projects, many new features and improvements could be implemented. As a result, this iteration is going to be a part of a series of development cycles. In this section, some of the most valuable proposals for the evolution of the system are outlined.

First of all, the user accounts functionality has to be extended. User accounts should allow users to save their sessions for future access. This can be done using different approaches, some of which are:

- *(Recommended)* Change the landing (initial) page to a choice of either login in or using the system as a guest. Allow logged in users to save multiple sessions, while guest sessions are cleaned.
- *(Moderate)* Only allow registered users to use the system, thus no need to handle anonymous guest sessions.
- *(Easy)* Each user can be connected to a single session and the user account is used for the purpose of retaining that session.

Also, user accounts can be used to restrict access to the system by only allowing users that hold an academic email for example.

It is important to note that while inspecting the inherited system, it was discovered that the system differentiates sessions using the users' *IP address*. This practice is wrong for many reasons, some of which are:

- Multiple users under the same NAT firewall or proxy will be identified as the same user by the system.
- A user might have a dynamic IP address which potentially could change during a session, thus losing all progress made.
- Users cannot open multiple sessions using different tabs or devices that have the same IP address.
- An attacker may be able to spoof an IP address and take over a session.

The standard way of supporting anonymous sessions is with the use of *cookies*. Cookies contain the session ID that the users' browsers store on behalf of the server. Every time a browser makes a request, it also sends that cookie so that the server can allow persistence between requests. This aspect of the system wasn't changed during this

iteration because it was thought that it is closely related to the approach chosen by the future developer for the user accounts functionality. If the system is only allowing registered users, then there is no need to implement anonymous sessions.

If the development of the system will support anonymous sessions then it is recommended to create a script that clears the files saved for these sessions and have it run using a job scheduler (usually referred as *cron* job). This is important because there is no clearing mechanism in place for inactive sessions and their files can add up wasting a lot of space on the deployed server.

Another suggestion is to deploy the site behind *HTTPS*, encrypting the connection for better security. Currently, the connection between the users and the back-end server is not encrypted, thus malicious network users could sniff or alter authentication credentials or any other information (e.g. the submissions folder) send in either direction.

The percentage similarity threshold introduced to make the results consistent, can be adjusted by the user using a slider (Sections 3.3.1 & 4.1). This ability to choose any value could be intimidating to users and thus they would leave the default value in place. Instead there could be for example three choices: *strict*, *moderate*, *flexible* that are some predetermined values deduced from additional testing of the similarity threshold parameter on different datasets.

The only available similarity detection package in the current system is *JPlag* [49]. The system was built to support multiple similarity detection packages. As discussed in Section 2.3.2 each of those packages has its own advantages and can be better for different cases. Hence, it is suggested that more similarity detection packages are included as an option. As discussed in Section 2.3.2, *MOSS* [17] supports some of the programming languages used in the School of Informatics that other packages don't support, making it a relatively good choice among other available packages. Note that *MOSS* is only available as an online service and thus will not run locally on the virtual server.

The system can be expanded to automate the process of the investigation done before having to utilise the Misconduct Assessment Tool. More specifically, automate the steps taken to detect similarities between submissions *pairwise*. Using a similarity detection package like *MOSS* [17] usually involves executing a script with different parameters manually. The system could instead through its interface receive all of the submission files and determine the parameters which the detection packages need to be executed with. Then it could run the detection for multiple packages in batch. The results would be interpreted similarly to how they are interpreted from JPlag in the current system and displayed in a summarised form. Hence, it gives a more general view of the similarities by crosschecking the values from the different packages.

Finally, when the system is going to be used by the rest of the academic staff in the School of Informatics, the production deployment should be using a production level web server like Apache [1] or Nginx [9] (Section 3.5). Using a production level web server means that the site is served faster and more efficiently for reasons like optimising the delivery of static files. It also makes the system more secure by not exposing debug and other private information.

Bibliography

[1] “Apache HTTP Server Project.” [Online]. Available: https://httpd.apache.org/

[2] “Desktop Screen Resolution Stats Worldwide.” [Online]. Available: http://gs.statcounter.com/screen-resolution-stats/desktop/worldwide

[3] “JPlag - Detecting Software Plagiarism.” [Online]. Available: https://jplag.ipd.kit.edu/

[4] “jQuery.” [Online]. Available: https://jquery.com/

[5] “jsPDF - HTML5 PDF Generator.” [Online]. Available: https://parall.ax/products/jspdf

[6] “LaTeX A document preparation system.” [Online]. Available: https://www.latex-project.org/

[7] “Microsoft word.” [Online]. Available: https://products.office.com/en-gb/word

[8] “MySQL.” [Online]. Available: https://www.mysql.com/

[9] “Nginx - high performance load balancer, web server, reverse proxy.” [Online]. Available: https://www.nginx.com/

[10] “Stack Overflow Developer Survey 2018.” [Online]. Available: https://insights.stackoverflow.com/survey/2018/

[11] “The Colour Blind Awareness organisation.” [Online]. Available: http://www.colourblindawareness.org/

[12] “Django,” 2005. [Online]. Available: https://djangoproject.com/

[13] “ISO 32000-2:2017 - Document management - Portable document format - Part 2: PDF 2.0,” July 2017. [Online]. Available: https://www.iso.org/standard/63534.html

[14] “Anaconda Software Distribution,” Dec 2018. [Online]. Available: https://anaconda.com

[15] “Code of student conduct,” Oct 2018. [Online]. Available: https://www.ed.ac.uk/academic-services/staff/discipline/code-discipline

[16] “Java,” May 2018. [Online]. Available: https://www.java.com/en/

[17] A. Aiken, "MOSS (Measure Of Software Similarity) plagiarism detection system," 1998. [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[18] H. Berghel and D. L. Sallach, "Measurements of program similarity in identical task environments," *SIGPLAN Notices*, vol. 19, no. 8, pp. 65–76, 1984. [Online]. Available: https://doi.org/10.1145/988241.988245

[19] B. Beth, "A comparison of similarity techniques for detecting source code plagiarism," 2014.

[20] A. Bushway and W. R. Nash, "School cheating behavior," *Review of Educational Research*, vol. 47, no. 4, p. 623632, 1977.

[21] G. Cosma and M. Joy, "Source-code plagiarism: A uk academic perspective," 2006.

[22] J. L. Donaldson, A.-M. Lancaster, and P. H. Sposato, "A plagiarism detection system," *SIGCSE Bull.*, vol. 13, no. 1, pp. 21–25, Feb. 1981. [Online]. Available: http://doi.acm.org/10.1145/953049.800955

[23] G. Dumpleton, "mod_wsgi." [Online]. Available: https://modwsgi.readthedocs.io/en/develop/

[24] S. Engels, V. Lakshmanan, and M. Craig, "Plagiarism detection using feature-based neural networks," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '07. New York, NY, USA: ACM, 2007, pp. 34–38. [Online]. Available: http://doi.acm.org/10.1145/1227310.1227324

[25] T. Gallwey, "Plagiarism or coincidence: Which?" *The Irish Monthly*, vol. 7, p. 312319, 1879.

[26] J. Gao, H.-S. J. Tsao, and Y. Wu, *Testing and quality assurance for component-based software*. Artech House, 2003.

[27] Google, "JavaScript Code Prettifier," Mar 2019. [Online]. Available: https://github.com/google/code-prettify

[28] B. Grot, "Informatics 2C - Introduction to Computer Systems (INFR08018)." [Online]. Available: http://www.drps.ed.ac.uk/18-19/dpt/cxinfr08018.htm

[29] D. Grune and M. Huntjens, "Het detecteren van kopien bij informatica-practica," *Informatie (in Dutch)*, vol. 31, no. 11, p. 864867, 1989. [Online]. Available: https://dickgrune.com/Programs/similarity_tester/

[30] J. Hage, P. Rademaker, and N. van Vugt, "A comparison of plagiarism detection tools," 2010.

[31] A. Harley, "Icon usability," Jul 2014. [Online]. Available: https://www.nngroup.com/articles/icon-usability/

[32] M. Heon and D. Murvihill, "Program similarity detection with checksims," Major Qualifying Project Report, Worcester Polytechnic Institute,

April 2015. [Online]. Available: https://web.wpi.edu/Pubs/E-project/Available/E-project-043015-122310/unrestricted/CheckSims.pdf

[33] D. R. Hipp, "SQLite." [Online]. Available: https://www.sqlite.org/

[34] B. Jenny, "Color Oracle." [Online]. Available: https://colororacle.org/

[35] M. Joy and M. Luck, "Plagiarism in programming assignments," *IEEE Trans. on Educ.*, vol. 42, no. 2, pp. 129–133, May 1999. [Online]. Available: https://doi.org/10.1109/13.762946

[36] C. J. Kacmar and J. M. Carey, "Assessing the usability of icons in user interfaces," *Behaviour & Information Technology*, vol. 10, no. 6, pp. 443–457, 1991. [Online]. Available: https://doi.org/10.1080/01449299108924303

[37] K. Kalorkoti, "Informatics 2B - Algorithms, Data Structures, Learning (INFR08009)." [Online]. Available: http://www.drps.ed.ac.uk/18-19/dpt/cxinfr08009.htm

[38] T. Kohonen, *Self-Organization and Associative Memory / T. Kohonen.*, 01 1984.

[39] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, Aug. 1988. [Online]. Available: http://dl.acm.org/citation.cfm?id=50757.50759

[40] T. Lancaster and F. Culwin, "Classifications of plagiarism detection engines," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 4, no. 2, pp. 1–16, 2005. [Online]. Available: https://doi.org/10.11120/ital.2005.04020006

[41] J. M. Lang, *Cheating lessons: learning from academic dishonesty*. Harvard University Press, 2013.

[42] M. Novak, D. Kermek, and M. Joy, "Calibration of source-code similarity detection tools for objective comparisons," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2018, pp. 0794–0799.

[43] U. D. of Commerce, N. I. of Standards, and Technology, *Secure Hash Standard - SHS: Federal Information Processing Standards Publication 180-4*. USA: CreateSpace Independent Publishing Platform, 2012.

[44] K. J. Ottenstein, "An algorithmic approach to the detection and prevention of plagiarism," *SIGCSE Bull.*, vol. 8, no. 4, pp. 30–41, Dec. 1976. [Online]. Available: http://doi.acm.org/10.1145/382222.382462

[45] M. Otto and J. Thornton, "Bootstrap." [Online]. Available: https://getbootstrap.com/

[46] J. Patrzek, S. Sattler, F. van Veen, C. Grunschel, and S. Fries, "Investigating the effect of academic procrastination on the frequency and variety of academic

misconduct: a panel study," *Studies in Higher Education*, vol. 40, no. 6, pp. 1014–1029, 2015. [Online]. Available: https://doi.org/10.1080/03075079.2013.854765

[47] J. Plekhanova, *Evaluating web development frameworks: Django, Ruby on Rails and CakePHP*, Sep 2009.

[48] B. Prado, K. Bispo, and R. Andrade, "X9: An obfuscation resilient approach for source code plagiarism detection in virtual learning environments," in *Proceedings of the 20th International Conference on Enterprise Information Systems - Volume 1: ICEIS,*, INSTICC. SciTePress, 2018, pp. 517–524.

[49] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding Plagiarisms among a Set of Programs with JPlag," *J. UCS*, vol. 8, no. 11, p. 1016, 2002. [Online]. Available: https://doi.org/10.3217/jucs-008-11-1016

[50] S. Psomas, "The five competencies of user experience design," Nov 2007. [Online]. Available: https://www.uxmatters.com/mt/archives/2007/11/the-five-competencies-of-user-experience-design.php

[51] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "Similarity of source code in the presence of pervasive modifications," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Oct 2016, pp. 117–126.

[52] D. Sannella, "Informatics 1 - Introduction to Computation (INFR08025)." [Online]. Available: http://www.drps.ed.ac.uk/18-19/dpt/cxinfr08025.htm

[53] A. Santella, "The art of seeing: visual perception in design and evaluation of non-photorealistic rendering," 03 2019.

[54] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, and N. Elmqvist, *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2018.

[55] A. Sinnel, K. Vnnen-Vainio-Mattila, S. Kujala, V. Roto, and E. Karapanos, "UX Curve: A method for evaluating long-term user experience," *Interacting with Computers*, vol. 23, no. 5, pp. 473–483, 07 2011. [Online]. Available: https://dx.doi.org/10.1016/j.intcom.2011.06.005

[56] M. S. Turan, E. B. Barker, W. E. Burr, and L. Chen, "Sp 800-132. recommendation for password-based key derivation: Part 1: Storage applications," Gaithersburg, MD, United States, Tech. Rep., 2010.

[57] K. L. Verco and M. J. Wise, "Software for detecting suspected plagiarism: comparing structure and attribute-counting systems," in *Proceedings of the ACM SIGCSE 1st Australasian Conference on Computer Science Education, ACSE 1996, Sydney, NSW, Australia, July 1996*, 1996, pp. 81–88. [Online]. Available: https://doi.org/10.1145/369585.369598

[58] G. Whale, "Identification of program similarity in large populations," *The Computer Journal*, vol. 33, no. 2, pp. 140–146, 1990. [Online]. Available: http://dx.doi.org/10.1093/comjnl/33.2.140

[59] M. J. Wise, "Yap3: Improved detection of similarities in computer program and other texts," in *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '96. New York, NY, USA: ACM, 1996, pp. 130–134. [Online]. Available: http://doi.acm.org/10.1145/236452.236525

[60] Y. Xie, "Tool to assist with assessing evidence for academic misconduct," Master's thesis, University of Edinburgh, 2018.