MOSS GUI Project

Developing a Graphical User Interface for the MOSS Plagiarism Checker developed by Stanford University

Team RNG: Isaac Basque-Rice, [REDACTED], [REDACTED], [REDACTED], [REDACTED], [REDACTED]

CMP311: Professional Project Development & Delivery

BSc Ethical Hacking

2021/22

Section	***Team Member***
Introduction	Isaac Basque-Rice*
Executive Summary	[REDACTED]
Methods	[REDACTED]
Methods	[REDACTED]
Results	[REDACTED]
Discussion	[REDACTED]

Note that Information contained in this document is for educational purposes only.

*My sections in grey highlight

Executive Summary

BACKGROUND

Plagiarism is an incredibly prevalent issue in Academic institutions. Computing related courses are disproportionately affected by plagiarism and therefore the requirement of tools to check for plagiarism in code files submitted by students is incredibly important. One such tool is the MOSS (Measure of Software Similarity) tool developed by Professor Alex Aiken at Stanford University in California. MOSS, however, does have some issues which are not ideal for a busy lecturer to deal with. Firstly, files must be submitted to MOSS through a command-line script – making submission of files slow and tedious. Secondly, MOSS will only accept code files as submissions. This means that the lecturer must first spend considerable time and effort to unzip multiple layers of .zip archives containing student submissions which are downloaded from their university's VLE (Virtual Learning Environment). Thirdly, the time taken for MOSS to respond with results of a submission varies wildly meaning that the process is halted while the lecturer waits for a response from MOSS each time files are submitted. Our client, Dr Suzanne Prior, an introductory programming lecturer at Abertay University, finds the current methods of using MOSS to check for plagiarism tedious and time consuming and has therefore asked our team to develop a graphical user interface application which will automate the aforementioned preparation of files and submission of those files to the MOSS server. To achieve this, our application incorporated the following requirements:

- Use existing MOSS tool as back end
- Create a simple, easy to use graphical user interface
- Automate the unzipping of files downloaded from the Abertay VLE – MyLearningSpace
- Rename files to ensure identifiability
- Submit files to MOSS automatically
- Display results for ease of user review

BENEFITS TO OUR CLIENT

Our application brings many benefits to our client. Firstly the tedious and time-consuming task of extracting several layers of .zip archives has been completely eliminated from our client's workload. Secondly, when results are received from MOSS they are displayed in an easy to read display which allows the client to see a percentage similarity score and view individual pairs of files for manual comparison if required. Thirdly, as all files are submitted automatically, the client can essentially 'set and forget' – that is to say that the client can submit the .zip archive downloaded from MyLearningSpace to our application and leave it to work in the background while they continue with other tasks. As a result of these points, the client now has a much reduced workload allowing her to spend time on more important tasks instead of wasting valuable time using MOSS in the traditional manner.

WHAT WE DID AND HOW

During the planning phase, the team produced a Gantt chart detailing a timeline of work to be completed. The project team used this to keep work on track. To accomplish all the required tasks, the project team split into four groups: front end, back end, cloud, and testing. The front-end team focussed on developing the main interface of the application ensuring it met the client's specification. The back-end team focussed on developing the main functional programming behind the application which facilitated the handling and extraction of files and MOSS submission. The cloud team focussed on migrating the locally functional version of the application to a cloud hosting provider which allows access to the final application from any internet enabled device with a web browser anywhere in the world. The cloud team also setup our domain – mossguiabertay.co.uk – which points to the cloud hosted application. The testing team focussed on ensuring that the application functioned as intended without errors and conducting a usability test with independent subjects to ensure that the application met the requirement of being simple and easy to use.

RESULTS AND CONCLUSIONS

Following all work completed by the project team, we have produced a fully functional application which meets the client's brief with the following key features:

- Simple and easy to use graphical user interface
- Automated extraction of .zip archives downloaded from MyLearningSpace
- Automated submission of code files to MOSS server
- Clear and logical display of results for ease of user review
- SSL certificate present on hosted application to ensure security

Further to this, our application was tested comprehensively using the "Grey Box" method which included the following key points:

Unit Testing

The programmers tested each code module as they were created to ensure they functioned as intended.

Integration Testing

Code modules were tested to ensure that they interacted with each other as intended.

System Testing

The overall system was tested to ensure complete functionality. Any bugs or errors were reported to the programming team and promptly fixed.

Performance Testing

The application was tested to ensure that it performed at an adequate pace ensuring that its function was greatly improved compared to using MOSS in the traditional manner.

Security Testing

The application's SSL certificate was checked using 'SSL Labs' to ensure security of the application and prevention of attacks e.g. man in the middle attacks.

Usability Testing

The application was shown to an independent group of people who each were tasked with using the application to ensure that the ease of use requirement had been met.

Compatibility Testing

The application was tested on multiple web browsers running under multiple operating systems to ensure that it functioned as intended on multiple platforms.

During the development, the team made use of the GitHub student developer pack which entitled us to credit with DigitalOcean allowing us to host the application for free. This cost would need to be met by the client for ongoing web hosting. However, the team have furnished the client with a version of the application which can be run locally allowing the client to test the application to ensure its functionality meets their expectations before expending the cost of maintaining the cloud hosting service. Similarly, during development the team purchased the domain – mossguiabertay.co.uk – the ongoing costs for which (£11.99/year) would need to be maintained by the client. If the client wishes to continue with cloud hosting, the project team is available to assist with setup. Enquiries should, in the first instance, be directed to our project manager – Isaac Basque-Rice at 1901124@abertay.ac.uk.

To conclude, the team is pleased with the outcome of this project. We are satisfied that we have produced an application which meets the project brief and requirements detailed by the client as well as implementing some additional features which the project team felt would be of benefit to the client.

Table of Contents

1 INTRODUCTION

1.1 BACKGROUND

Plagiarism in university courses is, and has always been, a major issue that requires addressing. A student presenting someone else's code as their own presents them with an unfair advantage over other students, whilst disregarding the standards that they should hold themselves to as a developer.

Universities in the UK's Russell Group collectively reported 3,721 cases of academic misconduct in the academic year 2016-17, of which over 2,100 were cases of plagiarism (Marsh, 2018). Stanford University in California reported 373 cases of "honor code violation", (i.e., academic dishonesty) over the period 1991-2001, of which 139 cases were relating to computer science, an incidence rate of 37%, despite the student body in that department only consisting of 6.5% of the overall enrolment (Roberts, 2002). This high rate of plagiarism, coupled with the fact that manually checking each code file submitted by hand for plagiarism is a near-impossible task for humans to complete, led to the creation of the MOSS (Measure of Software Similarity) plagiarism checking service.

MOSS is a server hosted in Stanford that "determines the similarity of programs" by checking a selection of code files submitted by the user, often a lecturer or assessor, against one another (Aiken, no date). The method of submission prior to the development of our solution was to unzip the files provided to an assessor by the virtual learning environment and attach them individually to a command line submission form, which sends them to the server to be checked and returns a result in web page format for analysis by the user.

With the specific workflow required of Abertay assessors, there is an extra layer of complexity that results in much frustration on their part. Firstly, due to the nature of the requirements for many submissions, there are several other files present in the directories submitted by students that will not be accepted by MOSS (such as .txt, .docx, and .pdf), these must be isolated and removed or simply not submitted for the submission to work, which further slows down the command line submission.

Additionally, when the files are retrieved from Abertay's VLE, MyLearningSpace, they come in a single large zip archive, which has smaller zip archives within it representing individual students' submissions, which in turn may have archives requiring extraction within them. Having to extract three layers of archives to access a single student's work is undoubtedly a bottleneck in the assessor's workflow and requires addressing.

Finally, due to the nature of the command line application, full or relative paths to each target file must be explicitly declared for the submission to work, i.e., "`Unit\ 2\ Assignment/Student\ A/Student\ A\ Submission/Assignment/main.cpp`" repeated possibly hundreds of times in a single submission, and if there are multiple code files in a single submission, each of those must be manually accounted for, also. This is blatantly inefficient, and any method of mitigation is sorely needed.

The original Moss submission file is available in Perl only, however there are several clients provided by third party contributors that allow assessors to interface more comfortably with the

software. For example, a Python client (mosspy), an OCaml client, Java, PHP, and Nim versions of the submission form, as well as a Swing, Python, and WPF Graphical User Interfaces. Issues with these implementations are numerous, however, as all the non-GUI versions of the software suffer from similar or identical issues as the original submission method (inefficiencies, requirements to perform significant amounts of individual labour, etc.), and the GUI versions have their own issues, from being only semi-implemented with numerous bugs present and little to no updates (cacticouncil, 2017), to being Windows only (May, 2022).

The lack of an efficient method of checking for plagiarism in code is contrasted with the relative ease with which assessors can check for plagiarism with other forms of work, such as essays, due to the fact TurnItIn, a popular text plagiarism checker, is built in to MyLearningSpace. This has no doubt been a source of frustration for the client and her colleagues.

The client for this project, Dr Suzanne Prior, an introductory programming lecturer at Abertay University, found the process of using MOSS to be woefully inefficient due to the time-consuming nature of the process, and hence tasked a team of developers to create a graphical user interface to perform the actions required for submissions automatically.

1.2 Aim

The aims of this project were to create a GUI for the MOSS plagiarism checker software which takes C++ files, runs them through the checker, and outputs a result to the screen showing similarity between the various pieces of software submitted. This was to be done with a view to streamlining the plagiarism checking process as much as possible. To achieve this the project team aimed to tackle the three issues outlined in the above section without sacrificing any of the core features of MOSS.

The core principle the team developed under was to minimize user interaction with the MOSS checker to the greatest extent allowable. To this end, the team aimed to create a web application which accepted zip archives retrieved from MyLearningSpace, recursively extracted the archives within this larger archive, and submitted all C++ files found to MOSS in a single fluid motion from the user's perspective, with the largest remaining time-sink being the time it takes MOSS to process the files (which, incidentally, is not inconsiderable).

This approach solves two of the three issues simultaneously, both the unzipping issue and the path issue, by abstracting and automating both actions away from the user. To solve the third problem, this being the presence of invalid file types, some basic validation was implemented to check if a file ended in ".cpp" or ".h", the two file types seen most frequently in C++ programming, and discard any files which did not meet this criterion.

2 METHOD

2.1 METHOD

The Gantt chart created during the project proposal was used to deliver the Programming Education project. See Appendix B for the Gantt chart. The team had tasks planned out for each member from the beginning of the project development. Having planned which tasks each member of the team were to undertake helped greatly when beginning development and kept the team organised throughout the project. It was critical in the team's success in delivering this project. However, the login system was cut from development due to change in plans and time constraints once the team began programming the web app. User and Software manuals were two tasks the team hoped to achieve and deliver to the client. This was not possible due to unforeseen circumstances. Cutting these tasks from development freed up more time to focus on other tasks.

The group split into front-end, back-end, cloud and testing roles as planned. In the first weeks, the front-end team started with the landing page of the website whilst the back-end team programmed code to extract the zip file searching for C++ code and sending it to MOSS. The cloud engineer then set up hosting the project with a Digital Ocean droplet using the domain name mossguiabertay.co.uk. A droplet instance had to be used as internal server errors appeared when the code was functioning fine locally. The team initially used a GitHub repository to sync the project code to Digital Ocean but these errors stopped this helpful feature being used. Code had to then be manually uploaded each time to the Digital Ocean droplet.

In weeks 7 onwards the project was in its final stages, and the team began to run the web app on a nginx server. This caused many issues in the project as the programming code did not work with the cloud platform despite working locally. The cloud and programming team worked together to successfully fix the problems and integrate MOSSGUI on the cloud. Once the web app was hosted on the nginx server final checks were done by the programmers and cloud engineer to ensure core functionality of the web app.

Timescales were followed very well and the team finished all tasks on time the week of 15th April. Once the project milestones were met the tester began checking for any issues in the programmers and designer's code. Finishing tasks on time meant the tester was not rushed to check the product, enabling testing documentation to be created while testing was undertaken. Weeks 9&10 were dedicated to polishing the web app fixing any issues the tester identified, and fixes were implemented rapidly.

Extra features were implemented into the web app that were not listed in the Gantt chart. These are discussed in section 2.4 of the paper. During development the team decided adding these features would help the functionality of MOSSGUI and help further meet the client requirement of efficiency/ease of use when submitting programming code.

The Gantt chart helped create a plan for the team to follow but as discussed, this plan was changed a bit to either deliver the project on time or an issue caused development to halt, and workarounds were needed.

Incremental model

During this project we used an incremental model, see figure 1 for our incremental diagram. An incremental model was used because it is a flexible model that could allowed the MOSS GUI team to quickly adapt to unforeseen circumstances that resulted in some tasks being cut from the project such as the login system. Another reason it was chosen was to make the project easier to test and debug any issues. This model allowed us to identify improvements that could be made to our project after each step of the Gantt chart was completed. Using the incremental model allowed the project team to follow our timescale for each task because each stage was quickly passed onto the correct team member to be reviewed or to allow issues to be fixed. This meant that tasks were implemented into the cloud version with only minor issues mostly caused by time constraints, this was done to allow each stage of the project to be completed without any large errors. This method also allowed errors to be found and solved quickly during the development of this project so that the project's key features were completed to a high standard and within our timescale. More detail on the core features of the MOSSGUI project can be found in section 2.3 of this paper.

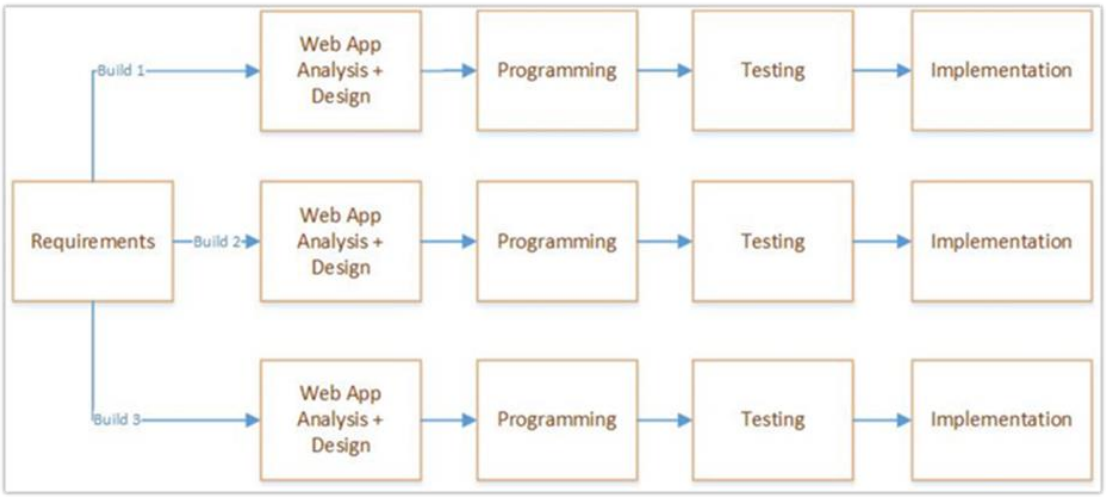


Figure 1 – incremental model diagram

2.2 TECHNICAL REQUIREMENTS

MOSSGUI had multiple requirements which were needed for the web app to function. They were:

- Python 3.9
- HTML
- CSS
- JavaScript
- Bootstrap
- Python flask
- Mosspy (soachishti, 2021)
- Domain name
- Digital Ocean droplet
- Nginx Server
- SSL certificate

2.3 CORE FEATURES OF MOSSGUI

To meet the client brief and ensure the product work as intended there were several key features the web app had to contain these features included the following:

- Creating a main page
- Designing a GUI (Graphic User Interface)
- Uploading a zip file
- Unzipping the zip file
- Displaying the results from the MOSS plagiarism checker
- The project must be secure

These features are discussed in more detail below

One of the main requirements of the MOSS GUI project was to create a product that is easy to use. To make sure our project was easy to use a webpage was created with a straightforward design, this can be seen below in figure 2. The webpage can be accessed by browsing to mossguiabertay.co.uk. On the main page, information to make uploading easier for users was added. This information included an explanation of what MOSS is with a link to a site containing more information. There are also instructions on how the MOSS GUI is used. These instructions explain what type of files are accepted by the MOSS GUI and explain that only 1 zip file can be uploaded at a time (although it can contain more zip files), this was done because the method used while unzipping files does not work with more than 1 zip file. Finally, the main page has a box to select files and upload them. Once selected the name of the chosen file is displayed. The colour scheme used was Gruvbox.

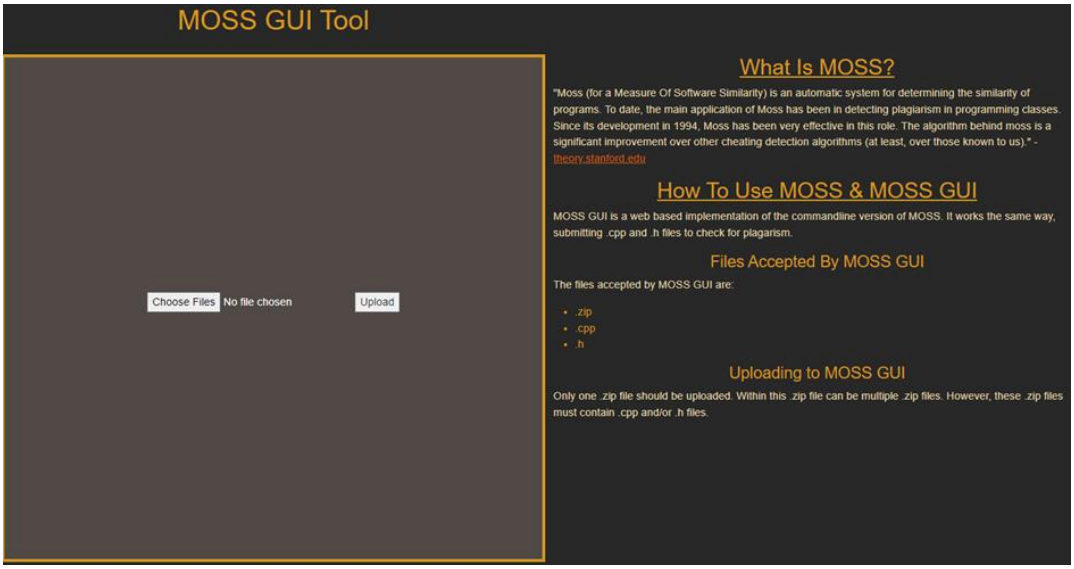


Figure 2 – MOSSGUI main page

To create the GUI, the open-source framework Bootstrap (version 5.1.3) was used because it is simple to use and contains many pre-built components and plugins useful for this project. The GUI contains HTML, JavaScript, and CSS. The GUI was designed with a mixture of the three. The main body of the GUI is created with HTML containing the text displayed on the screen and being used to call several functions and style sheets for the GUI. The style sheets of the website have been created in CSS and using the colour scheme Gruvbox. JavaScript was used within the design of the GUI as several functions were created in JavaScript. The use of the Bootstrap framework allowed for the creation of a GUI for the project to be completed within the project's timescale.

The MOSS GUI project also needed to allow a user to upload their files as discussed earlier, the main page of the MOSS GUI includes a button that allows the user to select a file and another to upload the selected file. To upload a file to the MOSS plagiarism, checker the user clicks the button that says “Choose files” this allows them to select a folder stored on their device as stated earlier this must be a one of the following: a zip file, .cpp file or a .h file. Once the file is selected. The name of the file is displayed on the screen. After this the user must click the button that says upload when this is done the function “uploadcheck.js” is used this will either display an error message this is shown in section 2.4 of the report. If the upload process was completed successfully then a message will be displayed to tell you the file has been successfully uploaded as shown in figure 3 below.

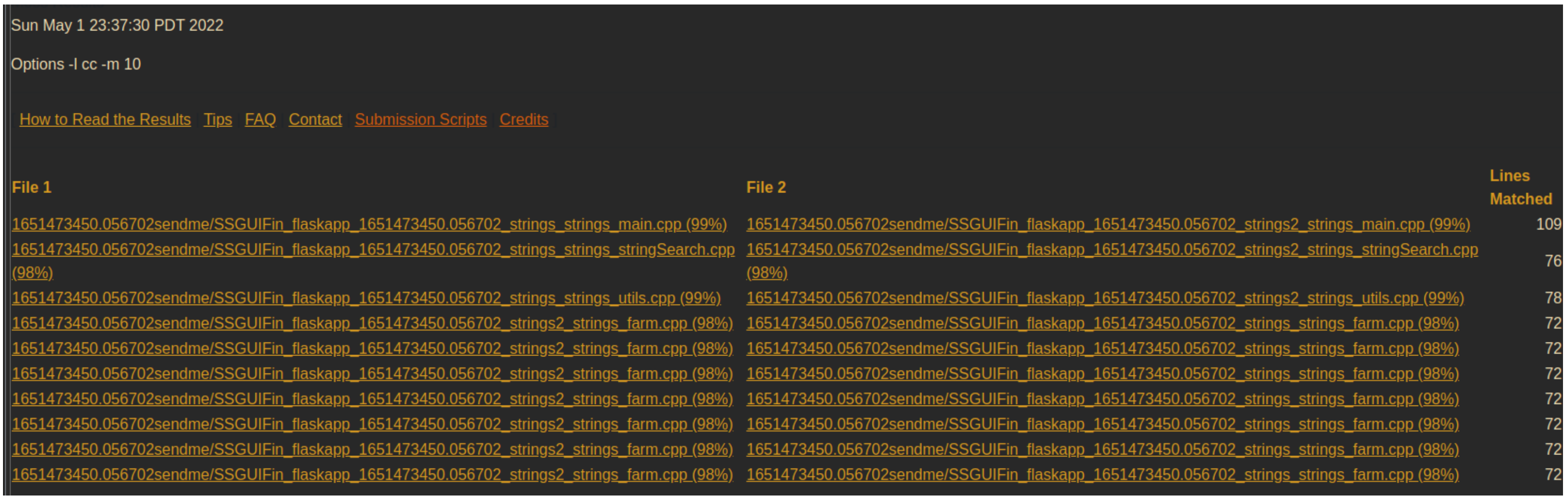
Figure 3 – MOSSGUI successful upload message

Once the file had been uploaded successfully it was unzipped to get all the .cpp and .h files, this function was implemented to the project by getting the file uploaded by the user, once this is done the program then extracts the archive. This may be done several times when a zip file contains more zip files within the uploaded file. After the archives are unzipped, directories are renamed adding the current date and time to the original directory names, ensuring the directory names are unique and allowing multiple directories to be stored at once. This was done to avoid having 2 directories with identical names. Once the directories have been renamed, individual files had to be found. This task was completed by searching through the directories for files using the file extensions .cpp or .h this ensured all c++ or header files were found. These files once found were all header and c++ files would be moved into a single directory. This directory is then sent to the plagiarism checker. This code was written in flask using python.

Python flask was a major component to the project (Pallets, 2022). This framework provided an easy method for the team to create the web app with python and implement the project. The upload feature on MOSSGUI was handled with flask and server requests were very easy to manage. Being able to manage server requests allowed the team to add custom error messages to the web app. Files uploaded to the flask app can be easily checked and the team made good use of this to save the files to the web app and implement validation checks.

MOSSGUI uses the mosspy library to submit the code to MOSS’s server for plagiarism (soachishti, 2021). This is a Python version of MOSS’s command line utility. Using a Python version of MOSS helped integrate it with python flask making it possible to host and run all the source code on a server. All the C++ and .h files are grouped together to send to MOSS with the mosspy functions. Mosspy simply returns a link of the results to the user like the Perl version. However, the objective was to increase efficiency. Therefore, the team implemented a method to display the report on MOSSGUI instead of a link.

Once a user uploads a .zip file to MOSSGUI the report containing the plagiarism results is automatically displayed. After MOSS returns the URL link of the report results to our server, MOSSGUI downloads and modifies it with custom CSS design. Figure 4 shows an example report displayed on MOSSGUI.



Sun May 1 23:37:30 PDT 2022

Options -l cc -m 10

How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits

File 1	File 2	Lines Matched
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_main.cpp (99%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_main.cpp (99%)	109
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_stringSearch.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_stringSearch.cpp (98%)	76
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_utils.cpp (99%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_utils.cpp (99%)	78
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72
1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings2_strings_farm.cpp (98%)	1651473450.056702sendme/SSGUIFin_flaskapp_1651473450.056702_strings_strings_farm.cpp (98%)	72

Figure 4: MOSSGUI plagiarism report.

MOSSGUI's report acts the same as the report returned from the command line Perl version of MOSS. Each link can be inspected for the comparison and code similarity. See Appendix C, figure 6. However, the custom CSS from MOSSGUI is not applied to these comparison pages as they as hosted on Stanford's website. The browser's previous page button can be used to simply return to the report after inspecting a link.

As security was a requirement of the client brief the team used certbot to obtain an SSL certificate for mossguiabertay.co.uk (DigitalOcean, 2021). See Appendix C, figure 7. The secure lock is displayed on the web app letting visitors know the traffic between user and server is protected with HTTPS. The SSL certificate was provided free by LetsEncrypt. As this free certificate expires after ninety days the team also set up auto-renewal and after thirty days a new SSL certificate will be requested. See Appendix C, figure 8.

2.4 Extra features of MOSSGUI

The team implemented more features that were not required in the brief. This improved the overall design and functionality of the web app and the experience for the user. These extra features include:

- **Error handling** – Checks if the user is using the web app incorrectly and stops the request.
- **Error messages -** Inform the user of how they are using the web app incorrectly. See Appendix C, figures 1-5.
- **RAR version** – Local RAR version of MOSSGUI "flaskappRAR.py" unzips and handles .rar files. Must be run in python3 from the terminal. The user may also have to install rarfile with pip. Will only run on Linux based OS. See Appendix C, figure 10.

3 RESULTS

3.1 RESULTS

3.1.1 Testing Results

The data used in this section was taken from the "MOSSGui Testing Documentation" document generated by the team's testers and can be found in appendix A – Testing Documentation.

The application was tested using the "Grey Box" testing technique, and it was split into the following points-

- Unit Testing
- Integration Testing
- System Testing
- Performance Testing
- Security Testing
- Usability Testing
- Compatibility Testing
- Documentation Testing

3.1.1.1 Unit Testing

The programmers performed the unit testing for each software package before pushing it to the main application. Each programmer tested their module and ensured it worked as intended and/or reported and communicated possible flaws to the team.

3.1.1.2 Integration Testing

The programmers performed the Integration testing after integrating a new module into the application. Integration tests were performed to ensure that the interaction between different modules worked together and produced an expected outcome.

3.1.1.3 System Testing

After the application was completed, the testing team performed a detailed analysis of the application to ensure that the application met all of the functional and client's requirements.

The testing team has split the client's requirements into the following points -

- Allow users to select and upload ZIP archives.
- Ensure that the user can upload only supported files.
- Find and unzip all files inside of the recursively zipped archive.
- Create a unique ID for each file to be submitted.
- Receive and display the results returned by MOSS.
- Check that all functionality enabling the above points works as expected.

The testers have found multiple issues within the application and reported their findings to the programming team. After the patches were applied to fix the initial problems, the application was tested again. Many issues were fixed during this step, but some were left due to time constraints.

3.1.1.4 Performance Testing

Our client's main objective and goal were to increase the efficiency with which the code similarities are detected compared to manual checking. To determine whether we have reached this goal, three volunteers and two team members were tasked with sending a zipped archive to the MOSS using our application. Their

attempts were timed, and in all five cases, the testers managed to upload the zipped archive in less than 30 seconds (waiting for MOSS servers to respond was omitted).

Next, the volunteers were tasked with manually preparing the files and sending them using "the previous method". In this test, the users struggled and needed around 10 minutes to find and unzip all the files in the zipped archive. Also, the time to manually prepare the file is directly correlated with the file sample size. However, the file sample size did not affect the time when using our application.

MOSS has crashed multiple times during the testing, making it difficult to perform tests with outside volunteers. Therefore, waiting for MOSS to respond was omitted as it was beyond our control.

3.1.1.5 Security Testing

As a web service, the security of our application is essential. The testing team has "security tested" our application by using an "SSL LABS". The SSL LABS has given our application an A rating. (The full report can be found at https://www.ssllabs.com/ssltest/analyze.html?d=MOSSguiabertay.co.uk)

3.1.1.6 Usability Testing

To test the usability of our application, the three volunteers were used to assess the UI elements of our application, especially the "user friendliness", "efficiency", and the "visual appeal". After the test, the users were asked to fill in the survey. As we can see from the graph in figure 5, the overall feedback from the survey was positive. However, most volunteers complained about the lack of information and visual updates after the files were submitted.

The volunteers were also asked to fill in the second survey to calculate the "System Usability Scale" (SUS). The SUS score was calculated, and the average SUS score added up to 71.66 points.

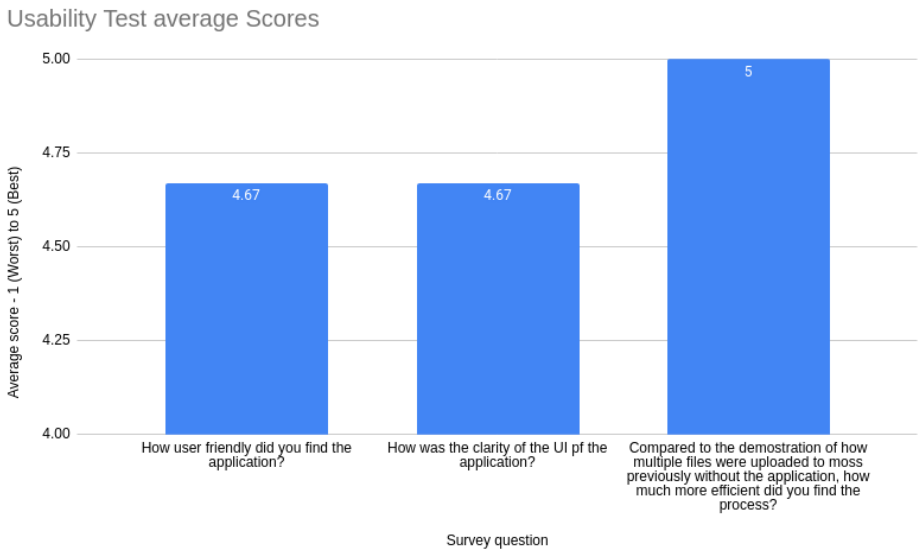


Figure 5 – Graph showing average score for questions asked in the survey

3.1.1.7 Compatibility Testing

As a web service, our application has to work on a wide range of devices, operating systems and browsers. The testing team has tested the application and confirmed that our application successfully runs on the following -

- Browsers
  - Chrome
  - Firefox
  - Safari

    - Internet Explorer
- Operating Systems
    - Windows
    - OSX
    - Linux
    - Android
- Devices
    - Laptop
    - Mobile
    - Desktop

3.1.2 Costs

During the development, our only costs were cloud hosting and domain name. The domain name "MOSSguiabertay.co.uk" was purchased for £4.99 for a single year and will have to be renewed for £11.99 each year.

Domain Name	Status	Expires On	Auto-renew	Estimated Value (USD)	Domain Listing Status	Protection Plan	Lock	Renewal Price
mossguiabertay.co.uk	Active	07/03/2023	On	Not available	Not Listed	None	On	£11.99/yr

Figure 6 – Domain name details

The "Digital Oceans" droplets were used to host our application. Our team has used the $5/mo offer for the development purposes, but once the final version of our application was released, it was upscaled to the $20/mo offer giving us 80 GB of disk space to provide faster service when handling multiple connections at once. In addition, as seen in figure 8, the $20/mo offer handled all the requests easily.

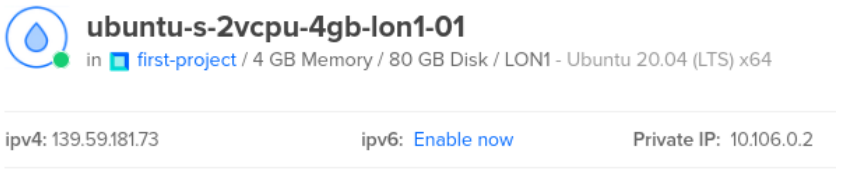


Figure 7 – Details of droplet hosting the application

Figure 8 – Graphs representing the bandwidth, CPU, and disk usage of the droplet

For the duration of the development, our team has used the free $100 trial offer. Therefore during the development, our team has spent only £4.99.

3.1.3 Security and exploits

Our team has done everything to make this application as secure as possible. Our application is hosted using a reputable hosting provider, "Digital Oceans". Also, our application makes use of an SSL certificate to prevent attacks like the "Man in the Middle" attack. And lastly, our application removes the uploaded files after the results from the MOSS are returned, which means that our application does not store any personal information making it GDPR complicit.

Currently, the application is functional. However, it contains some identified and most likely some undiscovered bugs. The following are the bugs discovered during the testing. However, due to time constraints, these were not fixed-

- The application ignores the WinRAR files.
- Some files are not sent to MOSS when selecting multiple ZIP (files not in a single archive).

4 DISCUSSION

4.1 GENERAL DISCUSSION

During the testing stage of the project, a range of methods were used to test various elements of the application. There were both positive and negative results from testing, but the project was an overall success. The content shown during the results section of this paper will be discussed in more detail below, to tie this back into the client's requirements. Figure 9 below displays the requirements that were agreed.

ID	List of Agreed Non-Functional Requirements	ID	List of Agreed Functional Requirements
1	Implement a solution that allows for increased user efficiency when submitting to moss	1	A serverside website hosted online
2	Improve the usability of the current MOSS application by improving the visuals of the app and decreasing the time taken to use it	2	Source Code for the website
3	Ensure the app is as simple as is feasible so as to not clutter the user's view and confuse them unnecessarily	3	A User manual to instruct the user and developer how to work with the code
4	Secure the sending and receiving of data to the best of our ability between the client and server, and the server and MOSS	4	A Requirements Specification
		5	Documentation regarding software used in this process and how to work with it for future developers
		6	Documentation regarding the testing process and how it was carried out

Figure 9 - Clients Requirements

Non-Functional Requirement 1:

During performance testing a significant decrease in time to use moss was recorded when using the MOSSGUI application. Considering this was based around a relatively small sample size of files compared to what the client expects, this shows the app will only increase in efficiency. Increased efficiency was a main goal of the project and was achieved.

Non-Functional Requirement 2:

Elements of this requirement links into the first about increasing efficiency but there is a distinct difference. The increased efficiency largely comes from the application automatically extracting and dealing with files contained in the ZIP archive. But the application also decreases time taken to use the MOSS service by providing a GUI (graphical user interface). This replaced the need to use a command terminal, with a much more intuitive and appealing interface. An average score of 4.67/5 for UI clarity from the volunteer testers confirmed the interface was visually appealing and functional.

Non-Functional Requirement 3:

For the GUI to truly help decrease time taken to use MOSS, it needed to be simple to use. The design is a simple two button layout, with instructions and on-screen prompts to keep the user informed. The volunteer testers confirmed it was simple to use and efficient, rating user friendliness 4.67/5. An average score of 71.66 for the SUS survey ensured the app held up to an industry standard review. A score exceeding 68 is generally considered above average (System Usability Scale (SUS) | Usability.gov, 2022).

Non-Functional Requirement 4:

The online version of the MOSSGUI application has SSL enabled and is A rated by SSL Labs for security (Qualys SSL Labs, 2022). Additionally, the application does not store data after it has been used, this ensures GDPR compliance (Guide to the General Data Protection Regulation, 2022). This covered the relatively small scope of security risks.

Functional Requirement 1:

The MOSSGUI application is a website hosted online via Digital Ocean (DigitalOcean – The developer cloud, 2022). This meets the conditions agreed to. The application is hosted at https://mossguiabertay.co.uk/.

Functional Requirement 2:

All source code for the project will be included during the handover to the client.

Functional Requirement 3:

Due to changes in team structure (loosing a member) this item of documentation was removed from the scope of the project to ensure completion of other elements. The simplicity and on-screen instructions ensured this would not be an issue.

Functional Requirement 4 & 5:

The source code of the project includes sufficient documentation and comments to cover both these requirements. All software used and requirements for operation are hardcoded into the source code to ensure accessibility for any user.

Functional Requirement 6:

The testing documentation can be found in "Appendix A – Testing Documentation" and will be included as a stand-alone document to the client during handover.

The MOSSGUI application meets all requirements agreed too, whilst being simple to use and access. This ensures using the application doesn't add to an already time-consuming process and simply makes plagiarism checking easier. The MOSSGUI application also stands out from other similar products by being web based. This provides the unique advantage of making it accessible anywhere, on any device with an internet connection. Additionally, the application is fully functioning. Alternative solutions don't offer these benefits, as mentioned during the introduction to this paper (cacticouncil, 2017), (May, 2022).

4.2 CONCLUSIONS

The MOSS GUI application offers the following benefits:

- Easy to use graphical user interface confirmed by both independent and industry standard surveys.
- Increased efficiency by automatically dealing with archived files.
- Local hosted version of application available.
- Work from any device with the online version of the application, no setup required.
- Online version securely implemented for peace of mind. Backed up by an A rating from SSL labs.

The MOSSGUI application offers the user an intuitive and efficient platform for submitting large files to the MOSS service. The application will significantly reduce the time taken to perform plagiarism checks, freeing up valuable time for other tasks. There is no downside to using the application, no training is required, easy setup and low/no-cost implementation is possible via the local version. Not using the application simply results in lower efficiency and makes deadlines harder to achieve. Currently the only web based and fully functioning interface for MOSS.

4.3 FUTURE WORK

Given more time for development there are several aspects that can be improved. At present the application only handles a few different file types. This can be increased to include a wider range of coding languages, so the application has a larger scope of operation.

The upload functionality also requires all files to be contained within a single .ZIP file, for easier use this can be amended to allow the selection of multiple files at once. This would further increase efficiency by decreasing setup time.

Additional features can be added to make the application more accessible. Such as audio descriptions for onscreen content, customizations for font, colour, and size. This would make the application easier to use for people with visual or hearing impairments.

Two addition functionalities that had been planned for the application but were removed due to time constraints were a drag and drop feature and a file listing menu. This would make it easier for users to submit files to the application and see what had been added.

4.4 CALL TO ACTION

To show how confident the project team are about the MOSSGUI application and its benefits. The locally hosted version of the app can be installed at no cost to the client. This would allow full access on one device, to all the benefits the application has to offer. Without the commitment, cost, and risk of purchasing the hosting services for the online version. Once satisfied the application can adequately perform its role, the project team can assist with a full setup. Training help is available, but due to its design is unlikely to be required.

Additionally, once in place the project team are available for support with all aspects of the application. All contact for support can be directed to the project manager Isaac Basque-Rice at: 1901124@abertay.ac.uk.

REFERENCES

Affairs, A.S. for P. (2013) *System Usability Scale (SUS)*. Department of Health and Human Services. Available at: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (Accessed: 30 April 2022).

Aiken, A. (no date) *Plagiarism Detection*. Available at: https://theory.stanford.edu/~aiken/moss/ (Accessed: 30 April 2022).

cacticouncil (2017) *Releases · cacticouncil/mistletoe*, *GitHub*. Available at: https://github.com/cacticouncil/mistletoe/releases (Accessed: 5 May 2022).

Guide to the General Data Protection Regulation (2018) *GOV.UK*. Available at: https://www.gov.uk/government/publications/guide-to-the-general-data-protection-regulation (Accessed: 1 May 2022).

Marsh, S. (2018) 'Cheating at UK's top universities soars by 40%', *The Guardian*, 29 April. Available at: https://www.theguardian.com/education/2018/apr/29/cheating-at-top-uk-universities-soars-by-30-per-cent (Accessed: 30 April 2022).

May, S.C. (2022) *shanemay/MossApp*. Available at: https://github.com/shanemay/MossApp (Accessed: 5 May 2022).

Qualys SSL Labs (no date). Available at: https://www.ssllabs.com/ (Accessed: 1 May 2022).

Roberts, E. (2002) 'Strategies for promoting academic integrity in CS courses', in *32nd Annual Frontiers in Education*. *32nd Annual Frontiers in Education*, pp. F3G-F3G. doi:10.1109/FIE.2002.1158209.

soachishti (no date) *mosspy: A Python client for Moss: A System for Detecting Software Similarity*. Available at: https://github.com/soachishti/moss.py (Accessed: 3 May 2022).

soachishti 2021, mosspy, *PyPI*, Available at: https://pypi.org/project/mosspy/ (Accessed: 3 May 2022).

DigitalOcean 2021, How do I install an SSL Certificate on a Droplet? :: DigitalOcean Documentation, *Docs.digitalocean.com*, Available at: https://docs.digitalocean.com/support/how-do-i-install-an-ssl-certificate-on-a-droplet/ (Accessed: 3 May 2022).

Pallets (no date), Welcome to Flask — Flask Documentation (2.1.x), *Flask.palletsprojects.com*, Available at: https://flask.palletsprojects.com/en/2.1.x/ (Accessed: 3 May 2022).

APPENDICES

APPENDIX A – TESTING DOCUMENTATION (WRITTEN BY [REDACTED])

```
# MossGui Testing Documentation - by [REDACTED]

The technique used for testing can be described as Grey Box Testing, Due to both
White Box and Black Box approaches being used to examine functionality. The
following methods were used:

| Functional          | Non-Functional |
| ----------          | ---------- |
| Unit Testing        | Performance Testing |
| Integration Testing | Security Testing |
| System Testing      | Usability Testing  |
|                     | Compatibility Testing |
|                     | Documentation Testing |

## Unit Testing -
As each section of code was completed by the programmers, this was tested by them
and the testing team. This was to ensure two modules of code were not combined and
underlying bugs/errors from one affected another. For example, when the code for
extracting data from a zip file was completed; this was tested before being
combined with the code for sending data to MOSS. This approach was used during the
coding of the application to minimise testing time towards the end of the project.
This was an ongoing process during development.


## Integration Testing -
When modules of code were combined with other sections of the application, testing
was once again performed to ensure these behaved as expected before further
sections were introduced. This was primarily performed by the programmers as they
developed the application with support of the testing team w needed.


## System Testing -
Once the programmers had finished developing the application, it was handed over
the testing team to perform an in-depth analysis to ensure all client requirements
had been met and functioned as intended. To achieve this several sub requirements
were derived from the following requirement agreed to by the client at the start of
development: Implement a solution that allows for increased user efficiency when
submitting to MOSS. To make sure this had been achieved the following were decided
upon.

- Allow user to select and upload zip file
- Ensure only supported files can be uploaded
- Ensure app extracts data from recursively zipped files
- Ensure app creates unique ids for each file to be submitted
```

- Retrieve and display results from MOSS after files have been sent
- check all functionally that enables the above works as expected

Using a Blackbox approach each of the above stated requirements were tested. This was a multistage process where the results were stored, so any issues could be re-created and assessed for repair by the programming team. Each phase of testing is documented below:

Phase 1 Testing Results

ISSUE	Recommendation
Drag and Drop feature doesn’t work.	Due to time constraints it is suggested this be removed from the application as its not a crucial element
Files are not listed on the right as suggested.	Due to time constraints it is suggested this be removed from the application as its not a crucial element
Error message is the same for MOSS down and incorrect file type.	Error messages should be amended to be unique as to give clarity to user about what is wrong
No progress message after submitting files to MOSS	A HTML tag can be added to be displayed on click to inform the user the request has been submitted and response from MOSS is required

Phase 2 Testing Results

ISSUE	Recommendation
Application doesn't like pure WinRAR Archives only WinRAR ZIP Archives when uploading	Check code used to determine supported files and ensure both .zip and .rar are included
If you select multiple ZIP files it doesn't appear to send the contents of both to MOSS	Due to time constraints, it is recommended to state all files should be within one archive. This means no additional code is needed
Doesn't allow you to upload .CPP files outside of a ZIP	This can be avoided by informing the user on archived files are accepted
Allows you to upload incorrect file type if done alongside a valid file	Ensure application only sends the .cpp/.h files when submitting the request to MOSS

Phase 3 Testing Results

This phase of testing was to ensure all major issues identified so far had been corrected and did not incur additional issues. After the third phase of testing no major issues were identified. All main concerns had been addressed and the requirements of the client appeared to have been satisfied, from the view of what was agreed. Minor spot testing will be carried out until completion, any issues raised will be documented alongside suggested solution within this document.

Performance Testing -

One of the main objectives for the project was to increase the efficiency for the user when uploading multiple files to MOSS. To determine if this was achieved both members of the testing team and three volunteer testers from outside of the project team were timed to gauge how long it took to upload the test files to MOSS using the application. In all five instances it was possible for the users to upload the test files within less than 30 seconds. To manually extract and upload the test files using the previous method, took in excess of 10minutes or longer depending on the user. The time taken to upload files does not noticeably increase as the sample size does when using the application. However, when manually uploading time spent directly increase in relation to the amount of files being sent. This clearly shows a significant improvement in efficiency. There is a slight increase in time when dealing with larger files and using the application, but not on a scale a human user would notice. This may vary slightly depending on the machine being used, as extracting time increase as the CPU power of the used machine decreases. The three volunteers used during testing have basic computing knowledge, this helping ensure the target audience should have no issues as they are considered experts within the field.

Security Testing -

Due to the nature of the application there isn't a great amount of security concerns. Most of the issues that could arise would be from the online hosting side of the application. Most of this is negated due to using a 3rd part hosting provider who incorporate most of the security as part of the fee for the service. However to ensure the hosting side of the application was secure and all user defined protocols were enabled, an SSL LABS security test was performed on the application. This gave the MossGUI application an A rank. This can be accessed from the following
link https://www.ssllabs.com/ssltest/analyze.html?d=mossguiabertay.co.uk.

Usability Testing -

As mentioned previously during the testing documentation, Three volunteer testers were used to assess various features of the application. This included how user friendly, visually appealing and efficient the application was during their test. The survey the volunteers completed also contain a section for addition comments if they had any. Each of the surveys from the volunteers are displayed below. Any issues raised by the volunteers were discussed and rectified with the team. In addtion to the survey below each volunteer performed the SUS survey, the score is attached to each of their files below.

Volunteer 1 - Ali
SUS Survey Score: 70
How user friendly did you find the application on a scale of, 1(not intuitive at all) to 5 (Very straight forward)?

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 4
- [ ] 5

How was the clarity of the UI of the application? On a scale of 1(unclear) to 5(very clear).

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 4
- [ ] 5

Compared to the demonstration of how multiple files were uploaded to moss previously without the application, how much more efficient did you find the process? On a scale of 1 (not faster) to 5 (significantly quicker).

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

Additional comments:
During the testing of the application, it failed to return the results and displayed an error message stating the service was not working. However, from the point of submitting the selected files there was no information displayed to say what was happening. It would be nice if there was some sort of progress bar or message stating that the application has finished and is waiting for a reply to the request.

Volunteer 2 - Lewis
SUS Survey Score: 75
How user friendly did you find the application on a scale of, 1(not intuitive at all) to 5 (Very straight forward)?

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

How was the clarity of the UI of the application? On a scale of 1(unclear) to 5(very clear).

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

Compared to the demonstration of how multiple files were uploaded to moss previously without the application, how much more efficient did you find the process? On a scale of 1 (not faster) to 5 (significantly quicker).

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

Additional comments:
Some text colour was hard to spot but other than that everything was clear enough to use. Nothing happens after hitting the upload button until you get the results, thats about all I can suggest everything else seemed to work fine.

Volunteer 3 (test performed after most issues had been addressed) - Liam
SUS Survey Score: 70
How user friendly did you find the application on a scale of, 1(not intuitive at all) to 5 (Very straight forward)?

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

How was the clarity of the UI of the application? On a scale of 1(unclear) to 5(very clear).

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

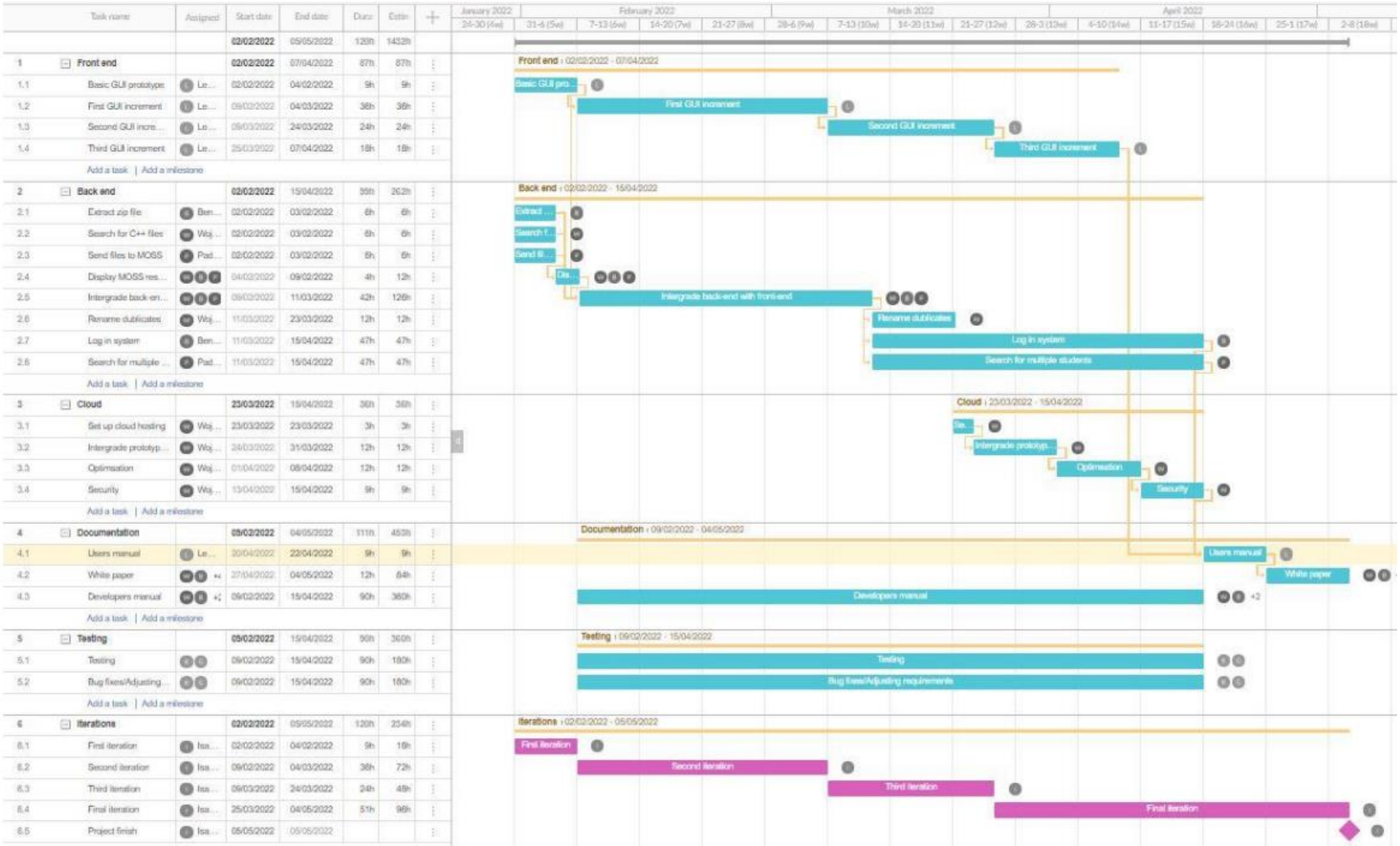
Compared to the demonstration of how multiple files were uploaded to moss previously without the application, how much more efficient did you find the process? On a scale of 1 (not faster) to 5 (significantly quicker).

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

Additional comments:
I had no previous experience using this service or application but i felt the interface was really straight forward and easy to use. My only thoughts is it felt

```
rather basic in terms of appearance but i guess that was maybe the point. I have
nothing to add to its functionality or instructions it was all really good.


## Compatibility Testing -
To ensure the application was compatible for the environment it would be intended
for, various browsers, devices and operating systems were tested to ensure
functionality. below is a list of environments that were tested successfully:

### Browsers
- Chrome
- Firefox
- Safari
- Internet Explorer

### Operating Systems
- Windows
- OSX
- Linux
- Android

### Devices
- Laptop
- mobile
- Desktop


## Documentation Testing
As part of the requirements agreed to with the client, various pieces of
documentation must be provided alongside the application itself. The following has
been checked for content, relevance and inclusion as part of handover.

- User manual
- Used/required software for application functionality
- testing documentation
- requirements specification
```

APPENDIX B – GANTT CHART



APPENDIX C – WEB APP FEATURES

Error Handling

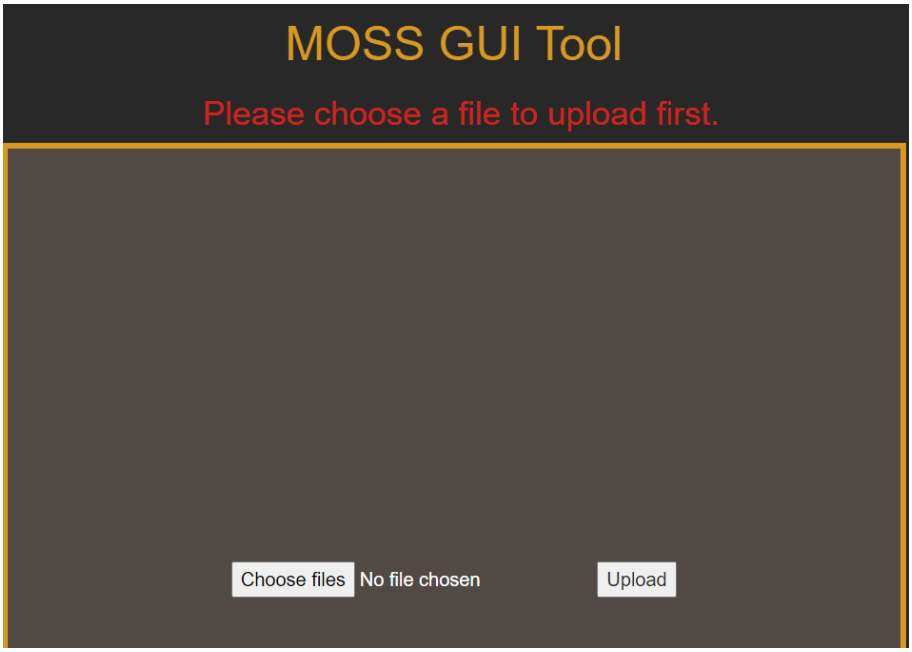


Figure 1: Error if no file chosen before uploading.

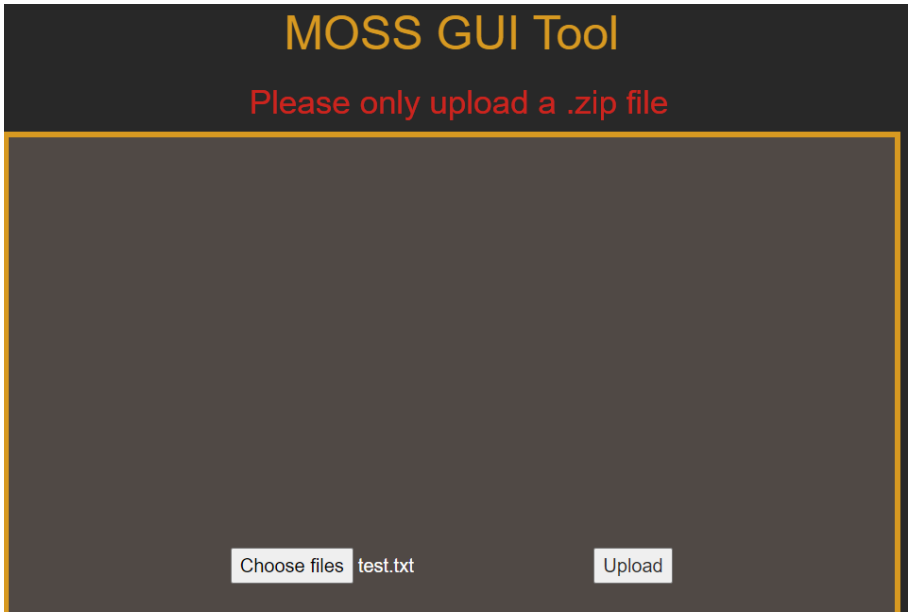


Figure 2: When trying to upload anything other than a .zip file.

Figure 3: When two .zip files are attempted to be uploaded.

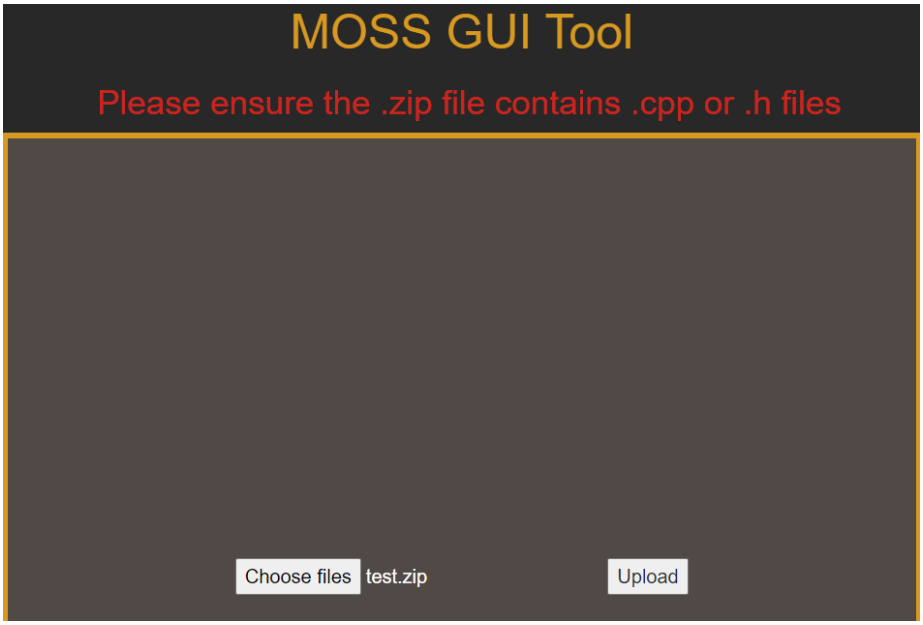


Figure 4: Error returns if the uploaded .zip file doesn't contain any code files.

Figure 5: MOSS servers are down.

MOSS Report

1651520064.505448sendme/1651520064.505448_testing_testing_strings_strings_main.cpp (99%)	[redacted]	1651520064.505448sendme/1651520064.50544 (99
26-134	[redacted]	26-134

1651520064.505448sendme/1651520064.505448_testing_testing_strings_strings_main.cpp

```
/*
Author: Patrick Collins
@license MIT
https://github.com/Paddylonglegs/
*/

#include <cassert>
#include <iostream>
#include <list>
#include <string>
#include <fstream>

//Threading
#include <cstdint>
#include <cstdlib>
#include <thread>

//Header Files
#include "utils.h"
#include "stringSearch.h"
#include "farm.h"
#include "task.h"
#include "channel.h"

// Import things I need from the standard library
[redacted]

using std::cout;
using std::endl;
using std::list;
using std::string;
using std::ofstream; //wrting timings to file
```

1651520064.505448sendme/1651520064.505448_testing_testing_strings2_strings_main.cpp

```
/*
Author: Patrick Collins
@license MIT
https://github.com/Paddylonglegs/
*/

#include <cassert>
#include <iostream>
#include <list>
#include <string>
#include <fstream>

//Threading
#include <cstdint>
#include <cstdlib>
#include <thread>

//Header Files
#include "utils.h"
#include "stringSearch.h"
#include "farm.h"
#include "task.h"
#include "channel.h"

// Import things I need from the standard library
[redacted]

using std::cout;
using std::endl;
using std::list;
using std::string;
using std::ofstream; //wrting timings to file
```

Figure 6: Comparison for plagiarism.

SSL Certificate

```
peter@ubuntu-s-2vcpu-4gb-lon1-01:/etc/nginx/sites-available$ sudo certbot --nginx -d mossguiabertay.co.uk
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator nginx, Installer nginx
Obtaining a new certificate
Deploying Certificate to VirtualHost /etc/nginx/sites-enabled/flaskapp

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: No redirect - Make no further changes to the webserver configuration.
2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for
new sites, or if you're confident your site works on HTTPS. You can undo this
change by editing your web server's configuration.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate number [1-2] then [enter] (press 'c' to cancel): 2
Redirecting all traffic on port 80 to ssl in /etc/nginx/sites-enabled/flaskapp

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Congratulations! You have successfully enabled https://mossguiabertay.co.uk

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=mossguiabertay.co.uk
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/mossguiabertay.co.uk/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/mossguiabertay.co.uk/privkey.pem
   Your cert will expire on 2022-07-09. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot again
   with the "certonly" option. To non-interactively renew *all* of
   your certificates, run "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

Figure 7: Obtaining SSL certificate for mossguiabertay.co.uk with certbot.

```
peter@ubuntu-s-2vcpu-4gb-lon1-01:/etc/nginx/sites-available$ sudo certbot renew --dry-run
Saving debug log to /var/log/letsencrypt/letsencrypt.log

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Processing /etc/letsencrypt/renewal/mossguiabertay.co.uk.conf
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Cert not due for renewal, but simulating renewal for dry run
Plugins selected: Authenticator nginx, Installer nginx
Renewing an existing certificate
Performing the following challenges:
http-01 challenge for mossguiabertay.co.uk
Waiting for verification...
Cleaning up challenges

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
new certificate deployed with reload of nginx server; fullchain is
/etc/letsencrypt/live/mossguiabertay.co.uk/fullchain.pem
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
** DRY RUN: simulating 'certbot renew' close to cert expiry
**          (The test certificates below have not been saved.)

Congratulations, all renewals succeeded. The following certs have been renewed:
  /etc/letsencrypt/live/mossguiabertay.co.uk/fullchain.pem (success)
** DRY RUN: simulating 'certbot renew' close to cert expiry
**          (The test certificates above have not been saved.)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

IMPORTANT NOTES:
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory will
   also contain certificates and private keys obtained by Certbot so
   making regular backups of this folder is ideal.
```

Figure 8: Auto-renewal set for SSL certificate.

Figure 9: SSL report from Qualys (Qualys SSL Labs, 2022).

RAR Version

```
┌──(kali㉿kali)-[~/Desktop/MOSS GUI Fin/flaskapp]
└─$ python3 flaskappRAR.py
 * Serving Flask app "flaskappRAR" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [02/May/2022 02:37:26] "GET / HTTP/1.1" 200 -
In RAR if
filecheck is testing.rar
File found: testing.rar
In send
before sending to MOSS
**********sent to moss
Report URL (Hosted on MOSS)http://moss.stanford.edu/results/5/701445749468
removing folder
Attempting to display report
Got past writing content
Stripped basename is 1651473450.056702.html
removing template report
127.0.0.1 - - [02/May/2022 02:37:32] "POST / HTTP/1.1" 200 -
```

Figure 10: Running RAR local version of MOSSGUI.

APPENDIX D - DELIVERABLES & REQUIREMENTS (REQUIRED)

Agreement Form: Project Deliverables

Group Name, Names of Team Members, and Programme	**TeamRNG** **Isaac Basque-Rice, [REDACTED], [REDACTED], [REDACTED], [REDACTED], [REDACTED]** **Ethical Hacking**
Subject specialist's Name (Client)	**Dr Suzanne Prior**
The deliverables listed below will be submitted by the team by the due date.	
Part A deliverables	**To be agreed by programme specialist and team, for example:** • Hosted Website • Source Code • User/reference manual • Requirements Specification, signed off by the programme specialist (see overleaf) • Software documentation • Testing documentation
Subject specialist's (Client) signature	
Team members' signatures	**IBRice** **[REDACTED]** **[REDACTED]** **[REDACTED]** **[REDACTED]** **[REDACTED]**

Agreement Form: Requirements

Group Name: TeamRNG

Team members (print): Isaac Basque-Rice, [REDACTED], [REDACTED], [REDACTED], [REDACTED], [REDACTED]

Project Title: MOSS GUI (Programming Education)

Please refer to the attached documentation for full details on the project. The requirements are listed in Table 1. The signatures below indicate that the requirements for this project have been agreed by the project stakeholders.

Any changes to the project documentation should be made using the correct change authorisation procedure agreed with the programme specialist.

Table 1

ID	List of Agreed Requirements (fill in)
1	Hosted Website
2	Source Code
3	User manual
4	Requirements Specification
5	Software documentation
6	Testing documentation

Stakeholders	Signatures	Date
Team members	IBRice [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]	14/02/2022
Subject Specialist		
Client (if applicable)		