Requirements Specification

for

COSC 4F00 Project

Version 4.4

Prepared by Cesar Moreno, Tamara McDiarmid, Saifa Bhanji, Levi Goertz, Andrew Folkerson, Curtis Penney

4F00 MOPP Analysis & Design

March 25th, 2019

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Project Scope 1

1.5 References 2

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Features 3

2.3 User Classes and Characteristics 3

2.3.1 Student 3

2.3.2 Instructor 4

2.4 Operating Environment 4

2.5 Design and Implementation Constraints 4

2.5.1 Hardware Limitations 4

2.5.2 Sandcastle Limitations 5

2.5.3 Security Considerations 5

2.5.4 Privacy Considerations 5

2.6 User Documentation 5

2.7 Assumptions and Dependencies 5

3. System Features 5

3.1 Local Repository: Authentication 5

3.2 Local Repository: Accept Student Submission of Assignment 6

3.3 Local Repository: Addition of Assignment to Repository 6

3.4 Local Repository: Retrieval of Course Submitted Information 6

3.5 Local Repository: Submission of Files to Remote Server 7

3.6 Local Repository: Submission Identifying Information Scrub 7

3.7 Local Repository: Submission Sending 8

3.8 Local Repository: Submission Receive Results 8

3.9 Local Repository: Submission Retrieve Results 9

3.10 Local Repository: Tokenization 9

3.11 Remote Server: Submission Analyzer 10

3.12 Remote Server: Returning Results 11

3.13 Front-End: Student Authentication 11

3.14 Front-End: Assignment Selection 12

3.15 Front-End: Group Assignment Submission 12

3.16 Front-End: Acknowledgement of Original Work 12

3.17 Front-End: Enter Username and Student Number 13

3.18 Front-End: Student Submission Upload 14

3.19 Front-End: Instructor Authentication 14

3.20 Front-End: Add Assignments to Courses 15

3.21 Front-End: Open and View Student Submissions 15

3.22 Front-End: Select Batch of Student Submissions and Submit for Analysis 16

3.23 Front-End: View Results of Analysis 16

3.24 Front-End: Auto-submit Assignments for Analysis 17

4. External Interface Requirements 18

4.1 User Interfaces 18

4.2 Hardware Interfaces 18

4.2.1 Supported Device Types 18

4.2.2 Supported File Types 19

4.3 Software Interfaces 19

4.3.1 Login Authentication 19

4.4 Communications Interfaces 19

4.4.1 Data Transfer Protocols 19

5. Other Nonfunctional Requirements 19

5.1 Performance Requirements 19

5.2 Safety Requirements 20

5.3 Security Requirements 20

5.4 Software Quality Attributes 20

6. Other Requirements 20

Appendix A: Glossary 20

Appendix B: Analysis Models and UI Mockups 21

Appendix C: Issues List 32

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Cesar Moreno | January 19, 2019 | Completed Introduction Section and Analysis Models | 1.0 |
| Cesar Moreno | January 23, 2019 | Added and reviewed System Features for Local Repository + System Features for Remote Server | 2.0 |
| Cesar Moreno | January 24, 2019 | Modified UML diagrams and added two more Use Cases | 2.1 |
| Cesar Moreno | January 25, 2019 | Removed Use Cases and System Features related to Teaching Assistant. Also highlighted all mentions of “Brock University Active Directory” in red for further review | 2.2 |
| Cesar Moreno | January 26, 2019 | Updated System Features for Local Repository | 2.3 |
| Cesar Moreno | January 30, 2019 | Reviewed System Features for Local Repository and Remote Server. Also completed Section 2 and added System Features for Front-End | 3.0 |
| Cesar Moreno | January 31, 2019 | Added Non-Functional Requirements for Section 5 + one new Use case | 3.1 |
| Cesar Moreno | February 1, 2019 | Reviewed UML diagrams | 3.2 |
| Cesar Moreno | February 1, 2019 | Added External Interface Requirements for User Interfaces in Section 4 | 3.3 |
| Cesar Moreno | February 3, 2019 | Included edits for Sections 1, 2, 3, 4, and 5 | 3.4 |
| Cesar Moreno | February 3, 2019 | Added mockup diagrams for UI in Appendix B. Also, added Hardware interfaces, Communication interfaces, and Software interfaces for Section 4 | 3.5 |
| Cesar Moreno | February 4, 2019 | Added minor edits to all sections | 3.6 |
| Cesar Moreno | March 13, 2019 | Added additional references in Section 1.5. Also, provided additional explanation in Section 2.5.1 of Hardware Limitations. Lastly, included an identification method for Submission Scrubbing in Section 3.6 | 3.7 |
| Cesar Moreno | March 20, 2019 | Added explanation of how Tokenization Algorithm works in Section 3.11. Also, modified Project Scope to reflect how the MOPP system changed during implementation in Section 1.4. Lastly, changed authentication method for Student and Instructor credentials in Section 3.14 and Section 3.20 | 4.0 |
| Cesar Moreno | March 21, 2019 | Modified Sequence Diagram “Instructor sends assignments to MOPP Remote Server” to reflect implementation changes in Appendix B. Also, added “Submission Listener Bug” issue in Appendix C | 4.1 |
| Cesar Moreno | March 23, 2019 | Changed authentication method for Local Repository in Section 3.1. Also, removed Submission Zipping functionality from Local Repository and assignment file zipping step in Section 3.5. Lastly, modified the storage method for Student and Instructor submissions in Local Repository in Section 3.2 and Section 3.3 | 4.2 |
| Cesar Moreno | March 24, 2019 | Reviewed Submission Analyzer algorithm description in Section 3.11. Also, changed authentication method used by MOPP system in Section 2.1 and Section 4.3.1. Also, removed Submission Zipping functionality in Section 3.23 and Section 3.25. Lastly, removed Section “3.20 Front-End: View Courses an Instructor Teaches” due to the MOPP system’s inability to SSH over Sandcastle. | 4.3 |
| Cesar Moreno | March 25, 2019 | Changed authentication method used by MOPP system in Section 5.3. Also, modified Use Case “Instructor selects batch of assignments in Local Repository” in Appendix B | 4.4 |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The “Measure Of Plagiarism Percentage” (MOPP) software has the purpose of checking students’ code submissions—from various computer science courses—for academic integrity. This implies that students must write and submit only their original work and respect university regulations. Due to the possibility that some students fail to adhere to these rules by using another person’s work without providing proper citation, MOPP employs a plagiarism-detection algorithm that creates an analysis of index of similarity with other students’ submissions. For the scope of this project, MOPP will only compare submissions written in Java, C, and C++ programming languages. Once the results are generated, the instructor will have the ability to view this analysis and compare submissions in a user-friendly interface. It is important to note that MOPP restricts the processing or divulging of the student’s personal information, thus any students’ submissions that are checked for academic integrity will be later erased from the MOPP server after the analysis of index similarity is created.

## Document Conventions

No document conventions have been determined thus far.

## Intended Audience and Reading Suggestions

This document is intended for Brock University’s instructor for COSC 4F00 and individuals who intend to utilize the software for academic purposes, such as other instructors and students.

## Project Scope

The MOPP system will be divided into a **local repository**—which stores students’ code submissions and that is located at the institution—and a **remote server**—where the analysis of index of similarity takes place and that is hosted by Brock University.

In regard to the MOPP Local Repository, a student has the capability of selecting the class and assignment number, and then upload his or her work as a compressed file (.zip). The submission will then be stored in an individual folder—identified by student login/student number—in this Local Repository, followed by a confirmation email being sent to the student to notify them of the submission’s success. Likewise, an instructor has the capability to log into the MOPP Local Repository in order to view the students’ submissions or submit a batch job to the MOPP Remote Server to perform an analysis resulting in an index of similarity. It is also possible to include student submissions from previous offerings of the course in the batch submission, since the instructor can upload old student submissions to the MOPP Local Repository. When it comes to the assignment submission to the MOPP Remote Server, after the instructor has selected the assignments that need to be analyzed, the MOPP Local Repository will first scan every submission in the selection and purge all personal information for confidentiality reasons, tokenize the scrubbed file, and finally transmit the tokenized file to the MOPP Remote Server.

In regard to the MOPP Remote Server, after the tokenized files are received, the Remote Server will apply the plagiarism-detection algorithm to tokenized files. This algorithm will compare student submissions against each other and create an index of similarity for each document. Once the algorithm has finished, it will store the indices of similarities, as well as any additional information required for the instructor’s analysis, within a results file that will be later sent to the MOPP Local Repository.

Upon request from the instructor, the MOPP Local Repository can then display the analysis results on the Front-End user-interface. This allows the instructor to judge whether a student has breached the academic integrity regulations of the university or not.

It is important to note that the MOPP software is intended to only receive and compare student submissions written in Java, C, and C++ programming languages.

## References

A. Jayapal, “Similarity Overlap Metric and Greedy String Tiling at PAN 2012: Plagiarism Detection”Department of Computer Science, University of Sheffield, UK, 2012

C. Kustanto and I. Liem, “Automatic Source Code Plagiarism Detection” ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed

Computing, pp. 481 – 486, 2009

Karp, Richard M. and Michael O. Rabin, ‘‘Efficient Randomized Pattern-Matching Algorithms,” IBM, Journal of Research and Development 31(2), p. 249−260, 1987

M. J. Wise, “String Similarity via Greedy String Tiling and Running Karp-Rabin Matching,” Department of Computer Science, University of Sydney, Australia, 1993

M. J. Wise, “Neweyes: A System for Comparing Biological Sequences Using the Running Karp-Rabin Greedy String Tiling Algorithm,” Department of Computer Science, University of Sydney, Australia, Technical Report 463, 1993

M. J. Wise, “YAP3: Improved Detection of Similarities in Computer Programs and Other Texts,” SIGCSE’96, p. 130-134, 1996

Schleimer, Saul., S. Wilkerson, Daniel and Aiken, Alex.: Winnowing: Local Algorithms for Document Fingerprinting*.*http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf. 2003

Sommerville, Ian. Software Engineering. 10th ed. New York: Pearson, 2016

# Overall Description

## Product Perspective

The goal of the product is to provide a single easy to use platform where Instructors and Students can upload assignment submissions to as well as compare assignments in order to detect similarities.

The existing tool, MOSS (Measure of Software Similarity), is a system designed at the University of Stanford which is utilized to determine the similarity of programs. The system described in this document will replace the use of MOSS.

Sandcastle is an existing system in place which acts as a server for the Computer Science department at Brock University. This system runs on a Linux environment and it will be where the Local Repository physically stores the submitted assignment files. The system described in this document will compare the user’s the login information with a database table of MOPP users to authenticate both Students and Instructors.

## Product Features

The system is broken down into three sections each responsible for their own overarching task.

The Front-End is responsible for providing the users an easy to use interface for the task at hand as well as organizing the user input and the information shown to the user. This will be where all user interaction occurs as both Students and Instructors will interact with the system through the Front-End. The interface will be implemented either through an HTML webpage or an application GUI. Due to the requirement of the system being accessible from computers external to the university, implementation through HTML is expected to be the preferred solution.

The Local Repository is responsible for managing the information on behalf of the Front-End UI and the Remote Server. When a user attempts to log into the system from the Front-End, the Local Repository will validate their information by authenticating the user’s credentials with Brock University’s Sandcastle. Should it succeed, user authenticates successfully. The Local Repository then acts as a central location where information is stored. When the Front-End requires information about what courses to display or assignments can be selected, it queries the Local Repository to receive the information. The Local Repository will also be able to connect to the Remote server and upload assignments for processing. Additionally, the Local Repository will receive the analysis of index of similarity results from the Remote server once the assignment files have been processed.

The Remote Server is responsible for receiving the tokenized documents of code before processing. The Remote Server then takes the tokenized documents and employs the plagiarism-detection algorithm to produce an analysis of index of similarity. Once the analysis is complete, the Remote Server collects results into a single file and sends it back to the Local Repository.

## User Classes and Characteristics

The system will be used by two primary user classes, Students and Instructor. Students will use the system as a means of submitting their assignments for the courses they are enrolled in. On the other hand, Instructors will use the system as a means of viewing the submitted assignments and comparing them for similarities.

## 2.3.1 Student

**Frequency of Use**

Students will be infrequent users of the tool given that they will mostly use the tool at most once per month per course.

**Functions Used**

Students will utilize the Authentication and the Assignment Submission features of the system.

**Expertise Level**

Students require basic computer skills to use this tool.

**Security and Privilege Level**

Students will only have access to the Front-End to upload their assignments.

### 2.3.2 Instructor

**Frequency of Use**

Instructors will be semi-frequent users of the tool as they will utilize the tool whenever assignments are submitted to any of their courses.

**Functions Used**

Professors will utilize the Authentication, Assignment Submission, and Assignment Processing features of the system.

**Expertise Level**

Instructors will require intermediate computer skills to use this tool.

**Security and Privilege Level**

Instructors will have access to the Front-End to upload assignments (e.g. uploading assignments from previous offerings of a course). In addition, instructors will need permissions for the folder locations on Brock University’s Sandcastle server, where the assignments will be stored. This means that instructors need to have access to courses they teach in order to view submitted assignments. In the case other instructors require access to assignments from a different course, they must ask the administrator to grant permissions.

## Operating Environment

The system must be able to run on an average student’s computer (e.g. i5 or i7 series Intel Processor and 4GB of RAM). In addition, the system must be accessible from computers outside the university’s internal network. Lastly, the system must be able to run on Windows, Linux and MacOS environments.

The Local Repository and the Remote Server will both be running on a Linux environment located on Brock University’s Computer Science environment known as Sandcastle.

## Design and Implementation Constraints

### 1 Hardware Limitations

1. The application must be usable on an average computer (e.g. i5 or i7 series Intel Processor and 4GB of RAM).
2. The Local Repository must be run on Sandcastle, which is a Linux environment.
3. The Remote Server must run on Sandcastle, which is a Linux environment.

### 2.5.2 Sandcastle Limitations

1. Permissions on Brock University’s Sandcastle server are out of the development team’s control, as such any limitations set in place by system administrators will likely transfer to MOPP.

### 2.5.3 Security Considerations

1. Access to a course’s assignments must only be provided to Instructors teaching the course or Instructors with proper authorization from the administrator.

### 2.5.4 Privacy Considerations

1. In the Local Repository, assignments must be scrubbed of identifying information before transmission to the Remote Server

## User Documentation

Public GitHub Repository: <https://github.com/4F00MOPP/MOPP>

## Assumptions and Dependencies

No assumptions or dependencies have been identified thus far.

# System Features

## Local Repository: Authentication

3.1.1 Description and Priority

The Local Repository (LR) receives an authentication request from the Front-End with login credentials. If the credentials are successfully validated with the database table of MOPP users, the LR will notify the Front-End that the login was successful.

3.1.2 Stimulus/Response Sequences

Stimulus: Authentication request from the Front-End.

Response: Authenticates with the database table of MOPP users.

3.1.3 Functional Requirements

REQ-1: Authenticates with database table of MOPP users.

REQ-2: Displays success or failure message.

REQ-3: Logs user attempts.

## Local Repository: Accept Student Submission of Assignment

3.2.1 Description and Priority

After the Student completes the submission process through the Front-End, the LR takes the assignment from the Front-End and stores it in a folder for that given Student, which is identified by the student login/student number.

3.2.2 Stimulus/Response Sequences

Stimulus: Front-End assignment submission confirmation.

Response: LR stores the submitted assignment appropriately and responds to the Front-End with a success or failure message.

3.2.3 Functional Requirements

REQ-1: Receives the assignment from the Front-End.

REQ-2: Stores the assignment in a folder for that given Student for later use.

REQ-3: Organizes the repository by hierarchy.

## Local Repository: Addition of Assignment to Repository

3.3.1 Description and Priority

After the Instructor selects an assignment to add, the LR takes the assignment from the Front-End and stores it in an individual folder designated for the Instructor’s assignment submissions.

3.3.2 Stimulus/Response Sequences

Stimulus: Front-End assignment submission from Instructor.

Response: Stores the submitted assignment appropriately and responds to the Front-End with a success or failure message.

3.3.3 Functional Requirements

REQ-1: Provides the Instructor a method of adding assignments.

REQ-2: Receives the assignment from the Front-End.

REQ-3: Stores the assignment(s) for later use.

REQ-4: Organizes the repository by hierarchy.

## Local Repository: Retrieval of Course Submitted Information

3.4.1 Description and Priority

Front-End requests the submission information from the LR. The LR retrieves the list of submitted assignments from Brock University’s Sandcastle server. In this case, the student number will be used for identification. This also allows the Instructor to view how many students have submitted assignments thus far. In the event the professor does not have permission to view the course folder, the LR will notify the Front-End.

3.4.2 Stimulus/Response Sequences

Stimulus: Front-End request for assignment information from Instructor.

Response: LR responds with the information about what assignments have been submitted and by whom, if the professor has permission for the course.

3.4.3 Functional Requirements

REQ-1: Retrieves the list of submitted assignments.

REQ-2: Retrieves batches of files.

REQ-3: Retrieves files from different levels of the repository.

## Local Repository: Submission of Files to Remote Server

3.5.1 Description and Priority

The LR receives a list of files to submit to the Remote server from the Front-End. The files can be from different Course-Assignment combinations. The LR will then pre-process the assignments by removing personal information and then it will tokenize every assignment file. Lastly, The LR will send the tokenized files to the Remote server for processing.

3.5.2 Stimulus/Response Sequences

Stimulus: Front-End sends a list of assignments to submit for processing.

Response: Initiates the processing by performing an information scrub on each file. Then tokenizes all scrubbed assignment files.

3.5.3 Functional Requirements

REQ-1: Initiates and manages the submission process.

REQ-2: Receives a list of files from the Front-End.

REQ-3: Runs each file through the Scrubbing process.

REQ-4: Tokenizes the assignment files.

REQ-5: Sends the tokenized assignment files to Remote Server.

## Local Repository: Submission Identifying Information Scrub

3.6.1 Description and Priority

Student submissions are stored in a fashion which identifies the author (e.g. individual folders identified by student login/student number). The LR will then parse through the specified Assignment and scrub identifying pieces of data from the contents of the assignment. Any instances of the following will be removed: Student Number, Student Name, Student Email, and Telephone Numbers. The LR will also remove any unnecessary files such as images or compiled code files.

3.6.2 Stimulus/Response Sequences

Stimulus: Receives a file to scrub of information.

Response: Scrubs the file and notifies when complete.

3.6.3 Functional Requirements

REQ-1: Allows removal of specified strings

REQ-2: Removes Student Name.

REQ-3: Removes Student Number.

REQ-4: Removes Email addresses.

REQ-5: Removes Phone-Numbers.

REQ-6: Remove unnecessary files (e.g. images, compiled code files, etc.)

## Local Repository: Submission Sending

3.7.1 Description and Priority

The LR will take the tokenized files and connect to the Remote Server. Once a secure connection has been established, the LR will send the tokenized files to the Remote Server for processing.

3.7.2 Stimulus/Response Sequences

Stimulus: Receives a selection of files to send to the Remote Server.

Response: Opens the connection to the Remote Server and sends tokenized files.

3.7.3 Functional Requirements

REQ-1: Receives the tokenized files.

REQ-2: Initiates connection with the Remote Server.

REQ-3: Sends the tokenized files to the Remote Server.

REQ-4: Awaits confirmation of files received.

## Local Repository: Submission Receive Results

3.8.1 Description and Priority

The LR allows the Remote server to connect back and upload the results from the analysis of index of similarity to the LR.

This feature will not require a user present as it could happen at any point after the submission has been made. In addition, a results notification message will be sent to the Instructor that submitted the request.

3.8.2 Stimulus/Response Sequences

Stimulus: Receives a response from the Remote Server notifying that a request from earlier has been fulfilled. Response contains a file with the results.

Response: Stores the results file on the LR and notifies the Instructor that the analysis of index of similarity has been completed.

3.8.3 Functional Requirements

REQ-1: Receives results from the Remote Server in the form of a file.

REQ-2: Stores the file with an identifying key specific to the instance.

REQ-3: Notifies the Instructor that the process is complete.

REQ-4: Must be able to receive the file at any time from the Remote Server.

## Local Repository: Submission Retrieve Results

3.9.1 Description and Priority

Upon request from the Front-End, the LR will be able to retrieve the analysis of index of similarity results file and send it to the Front-End for display. Later, if the user decides to delete this file, then it will be removed from the Local Repository.

3.9.2 Stimulus/Response Sequences

Stimulus: Receives request from Front-End to view the results of analysis of index of similarity.

Response: Sends the stored results file to Front-End.

3.9.3 Functional Requirements

REQ-1: Retrieves the analysis of index of similarity result file and sends it to the Front-End.

REQ-2: Removes the results file from Local Repository upon request.

## Local Repository: Tokenization

3.10.1 Description and Priority

The tokenization feature takes an existing document of code and decomposes it into a sequence of symbols or tokens, which identify broad features of the language (e.g. variables, types, looping statements, etc.). The tokenization algorithm works by accepting a line of input, preprocessing the data (such as removing unnecessary white spaces), and then checking the line against a table of regular expressions which correspond to tokens—substitutions are then made to produce a tokenized line. These new tokenized lines are written to a file that can be used by the comparison algorithm to perform the analysis of index of similarity.

Tokenization as a feature is of medium priority as it drastically simplifies and improves the ability of the Remote Server analyzer to efficiently and accurately detect similarity across a wide array of documents.

3.10.2 Stimulus/Response Sequences

Stimulus: Receives a selection of assignments from the Local Repository.

Response: Tokenizes assignment files to be analyzed in Remote Server.

3.10.3 Functional Requirements

REQ-1: Parses all the possible variations of code in the diverse languages supported by the application to some predefined lexicon.

REQ-2: Sends the transformed document forward to the analyzer to perform the analysis of index of similarity.

## Remote Server: Submission Analyzer

3.11.1 Description and Priority

The analyzer feature consists of analyzing an already tokenized document against other tokenized documents to determine whether there are any significant similarities.

The main comparator algorithm makes use of two approaches to compare any number of tokenized documents against each other. First, the Greedy String Tiling algorithm is used to generate all substrings of the maximum length for each of the tokenized strings being compared. These substrings are then compared using Karp Rabin hashing to determine if there are any matches. These matches are then recorded for future reference. Following this, the maximum string length is then reduced, and this process is repeated. By this method, tokenized documents can be analyzed for possible instances of academic misconduct.

This feature is of high priority as it is paramount to the underlying goal of the MOPP software.

3.11.2 Stimulus/Response Sequences

Stimulus: Receives tokenized documents that need to be analyzed.

Response: Produces analysis of index of similarity by applying the plagiarism-detection algorithm on the tokenized documents.

3.11.3 Functional Requirements

REQ-1: Compares tokenized documents against each other in a reasonable period of time.

REQ-2: Detects instances of similarity in the documents and flag them as they occur.

REQ-3: Generates an index of similarity results file after the documents have been analyzed.

## Remote Server: Returning Results

3.12.1 Description and Priority

The results sending feature simply returns the results file of the analyzer back to the Local Repository.

Returning results is of high priority. The entire purpose of the MOPP system rests on the function of accurately checking for instances of similarity and communicating the results of this search to the individual who initiated the request. As such, this function is critical to the performance of the system.

3.12.2 Stimulus/Response Sequences

Stimulus: Analyzer sends a signal which tells the Remote Server that the results are ready to be returned.

Response: Results file is sent to the Local Repository through a secure connection.

3.12.3 Functional Requirements

REQ-1: Transmits the information from the Remote Server to the Local Repository while maintaining the integrity of the data being transmitted.

## Front-End: Student Authentication

3.13.1 Description and Priority

Students log into the MOPP system with their MOPP login credentials in order to submit their assignments for computer science classes they are taking.

This is a high priority feature since the system must keep track of students’ submissions to avoid anonymous submissions.

This is also time sensitive, as Students should only be able to submit work if they are computer science students at the university.

3.13.2 Stimulus/Response Sequences

Stimulus: When a Student enters his or her MOPP login credentials in the MOPP Front-End login page, the login information will be compared to a database table of users to verify the Student’s identity.

Response: Students are presented with a success page and gain access to the MOPP assignment submission page, if the authentication is successful.

3.13.3 Functional Requirements

REQ-1: Database table that stores the MOPP users’ login information.

REQ-2: Successful login attempts will display a success prompt.

REQ-3: Unsuccessful login attempts will display with a failure prompt.

REQ-4: Users who could not successfully log in will be given an opportunity to

retry.

## Front-End: Assignment Selection

3.14.1 Description and Priority

After successful authentication, Students should be presented with a way to indicate which assignment they are submitting.

This feature is a high priority because students need a way to specify which class and assignment their submission corresponds to so they can be grouped together for analysis by the MOPP system.

3.14.2 Stimulus/Response Sequences

Stimulus: Students select files to upload.

Response: Students are asked to indicate which class and assignment number they are submitting an assignment for.

3.14.3 Functional Requirements

REQ-1: Students must be able to make submissions to classes that are running that semester.

REQ-2: Students must only be able to submit assignments that have been set up by the Instructor through the Local Repository.

## Front-End: Group Assignment Submission

3.15.1 Description and Priority

Once students have indicated the assignment number and course code, they should be presented with an option to indicate if the assignment they are making a submission for is a group assignment.

This feature is a high priority because most upper year Brock University’s computer science courses require group submissions.

3.15.2 Stimulus/Response Sequences

Stimulus: Students selects the group assignment submission option.

Response: Front-End prompts the student to enter the group members’ information before submitting the assignment.

## Front-End: Acknowledgement of Original Work

3.16.1 Description and Priority

The Front-End requires students to agree with an honor pledge to acknowledge that they are submitting original work.

This is high priority as students should not be allowed to make submissions without first agreeing with the honor pledge.

3.16.2 Stimulus/Response Sequences

Stimulus: Student agrees with honor pledge.

Response: Front-End accepts student submission. If the Student does not agree with the honor pledge, the student submission is declined.

3.16.3 Functional Requirements

REQ-1: Submission attempts without agreeing with the honor pledge are rejected.

REQ-2: If rejected, the Front-End displays a message explaining that the Student must accept the honor pledge before submission.

REQ-3: When the Student accepts the honor pledge, the assignment submission is now permitted.

## Front-End: Enter Username and Student Number

3.17.1 Description and Priority

The Front-End prompts students to enter their Brock University’s username (e.g. ab12xy) and student number before they are able to submit their assignment to Local Repository. This personal information will eventually be scrubbed from the student submissions before being sent to the Remote Server for analysis of index of similarity.

This feature is a high priority because we need to ensure the privacy of the students using the system.

3.17.2 Stimulus/Response Sequences

Stimulus: Student attempts to submit individual/group assignment.

Response: Front-End prompts the user to the enter username and student number for individual/group members. If submission is successful, Front-End displays confirmation message.

3.17.3 Functional Requirements

REQ-1: Students will only be allowed to enter information for more than one student if they are submitting a group assignment.

REQ-2: Students will only be allowed to enter information for the maximum number of students that the Instructor specified for the group assignment.

REQ-3: Upon successful submission, Front-End displays confirmation message.

## Front-End: Student Submission Upload

3.18.1 Description and Priority

The Front-End allows students to upload a compressed file (.zip) to the Local Repository for the course code they entered.

This is a high priority feature as the student will require this function to submit course work for the given course.

It is also time sensitive, as the student should only be able to submit work prior to and up to the due date and time. The system should not accept late submissions, unless the Instructor allows it.

3.18.2 Stimulus/Response Sequences

Stimulus: Students is prompted to select an assignment file from their computer. Then, the system attempts to add the student assignment to the Instructor’s folder in the Local Repository.

Response: Student submission is accepted, if it has the correct format, and a confirmation message is displayed. Otherwise, the student submission is rejected, and an error message is displayed. In this case, the student will have to option to resubmit.

3.18.3 Functional Requirements

REQ-1: The Student can select a compressed file (.zip) from their computer for submission. The system will only allow submissions in .zip format, while all other file formats should be rejected.

REQ-2: If the student submission is rejected, the Student is notified then prompted to repeat the process. This notification should inform the Student of reasons an assignment file can be rejected (e.g. file type).

REQ-3: If the student submission is accepted, the Front-End displays a confirmation message to the Student.

REQ-4: Students must not be allowed to make submissions past the deadline.

REQ-5: Students must be allowed to make multiple submissions, up to the deadline. Hence, before a student submission is committed to the Instructor’s folder in the Local Repository, the system must determine if this is the first submission for this particular student or if it is a subsequent submission.

REQ-6: If subsequent submission, the MOPP system must overwrite the old assignment file in the Local Repository with the newly submitted file.

## Front-End: Instructor Authentication

3.19.1 Description and Priority

Instructors will use their MOPP login credentials in order to view submissions to assignments for computer science courses they are currently teaching.

This is a high priority feature since the Local Repository must prevent anonymous users from accessing student submissions in the Local Repository.

3.19.2 Stimulus/Response Sequences

Stimulus: When the Instructor enters his or her MOPP login credentials in the MOPP Front-End login page, the login information will be compared to a database table of MOPP users to verify the Instructor’s identity.

Response: The Instructor is presented with a success message and gains access to the Local Repository to view submissions, if authentication is successful.

3.19.3 Functional Requirements

REQ-1: Database table that stores the MOPP users’ login information.

REQ-2: Successful logins will display a success message.

REQ-3: Unsuccessful logins will display an error message.

REQ-4: After error message is displayed, Instructor will have the opportunity to retry login.

## Front-End: Add Assignments to Courses

3.20.1 Description and Priority

Instructors are able to set up assignments for each of their classes and assign a deadline date to each one. Once the Local Repository has determined that the assignment deadline has passed, no more submissions should be allowed from students.

This feature is a medium priority as Instructors could also use their discretion to allow late assignments, or simply disregard assignments that were received after the due date.

3.20.2 Stimulus/Response Sequences

Stimulus: Instructor selects a course to view.

Response: Instructor is given the option to add or remove assignments and associate a deadline with each assignment.

3.20.3 Functional Requirements

REQ-1: Instructors can only pick future dates for deadlines.

REQ-2: Students must be able to see the assignments the professor has set up for

their class.

## Front-End: Open and View Student Submissions

3.21.1 Description and Priority

This feature will be available to the Instructor of a course. The Instructor will be able to view all the student submissions within their course folder and open the files as needed. The Instructor of the course is the only one that can view the student submissions for the course.

3.21.2 Stimulus/Response Sequences

Stimulus: Instructor attempts to open and view student submission in the Local Repository.

Response: Open and display the student submission.

3.21.3 Functional Requirements

REQ-1: Provide functionality to authorized Instructors to view student submissions for their courses before prior to sending submissions to Remote Server for analysis.

## Front-End: Select Batch of Student Submissions and Submit for Analysis

3.22.1 Description and Priority

This feature will be available to the Instructor of the course. The Instructor will be able to select several student submissions, then the Local Repository will be tasked with scrubbing personal information and then tokenize each file before sending it to the Remote Server for analysis of index of similarity. Note that the Instructor can also include student submissions outside from the current course folder to the batch. Once this process is done, the tokenized files will be sent to the Remote Server.

3.22.2 Stimulus/Response Sequences

Stimulus: Instructor selects current student submissions from current course folder or student submissions from previous offerings of the course to bundle.

Response: Send a list of student submissions that the Local Repository need to prepare (i.e. scrubbing and tokenizing) before analysis. Once the files are ready, they are sent to the Remote Server.

3.22.3 Functional Requirements

REQ-1: Authorized Instructors must be able to open their repository for their course.

REQ-2: Authorized Instructors must be able to add additional student submissions

(e.g. student submissions from previous offerings of the course) to their

submission batch.

REQ-3: Instructors must be able to submit the batch of files, which will be scrubbed

and tokenized first, to the Remote Server for analysis of index of similarity.

## Front-End: View Results of Analysis

3.23.1 Description and Priority

When the Instructor attempts to view the analysis results file stored in the Local Repository, the Front-End creates a weblink to view the results in a secure website. This website will show a percentage of similarity between two or more submitted assignments. Additionally, the Front-End will highlight the portions of the code that are similar across multiple submissions.

3.23.2 Stimulus/Response Sequences

Stimulus: Instructor accesses Local Repository and attempts to read the analysis results file.

Response: Front-End creates a weblink with the results file and displays all student submissions at once along with a similarity score for the submissions and it will allow the Instructor to select any student submission to review it.

3.23.3 Functional Requirements

REQ-1: Secure website must be used to render analysis results in a user-friendly manner.

REQ-2: All student submissions will be displayed at once with a similarity score associated with them. In this way, the Instructor can glance at the results and only view the submissions that they feel need to be reviewed based on their score.

## Front-End: Auto-submit Assignments for Analysis

3.24.1 Description and Priority

When setting up assignment due dates, the Instructor has the option to let the system auto submit all Student submissions to the Remote Server for analysis once the deadline has passed. During this process, the Local Repository will automatically scrub all personal information and tokenize all assignment files before establishing a connection with the Remote Server.

This feature is a low priority because the Instructor can also log in to the Local Repository after the deadline and manually run the submission.

3.24.2 Stimulus/Response Sequences

Stimulus: Instructor selects auto-submit option in the Front-End.

Response: Local Repository scrubs personal information and tokenizes assignment files before finally submitting the files to the Remote Server for analysis of index of similarity.

3.24.3 Functional Requirements

REQ-1: Local Repository must have access to the local time.

REQ-2: Local Repository must be online and running at the time the assignment

deadline.

REQ-3: The Remote Server must be online at the time the assignment is due.

REQ-4: Two variables must be present for auto-submission to take place: the auto

submission check box must be true, and the date and time of auto

submission must be set.

REQ-5: Front-End notifies Instructor after the auto-submission is completed.

# External Interface Requirements

## User Interfaces

This system will have a GUI in the form of a webpage known as the Front-End. The webpage will be the primary point of access for the users of the MOPP system. The webpage will interact directly with the Local Repository. The webpage will not have any awareness of the Remote Server, where the analysis of index of similarity is being performed.

4.1.1 The user interface must be secure and reliable.

4.1.2 The user interface must be efficient in speed and ease of use.

4.1.3 Users must feel confident when using the interface and be able to easily understand the

concepts needed for navigating the interface.

4.1.4 Error messages in the user interface will be presented in neutral language and help the

user to understand why the error has occurred and how to correct it.

4.1.5 The user interface will highlight dangerous operations that could lead to severe errors.

4.1.6 The user interface will make sure that interactions in the system will not be

misinterpreted.

4.1.7 Unnecessary interactions from the user will be avoided.

4.1.8 When processing time is required of the system, the user interface will clearly

communicate this to the user. Moreover, processing on the server will not stop the user from interacting with the user interface.

4.1.9 Student confidentiality will be maintained as this information is only stored on the

Local Repository—which scrubs all personal information from student submissions before analysis—and never given to the Remote Server. This will be clearly communicated to the Student users.

4.1.10 Users will be able to log out of the system on every screen.

4.1.11 Users will be able to access the user documentation for every screen.

## Hardware Interfaces

### 4.2.1 Supported Device Types

The system will operate on Windows, MacOS and Linux devices. The system may be operational on other devices however full functionality is not guaranteed.

### 4.2.2 Supported File Types

The MOPP will import compressed files (.zip) and decompress them for storage in Local Repository. The compressed files can contain arbitrary document types. Moreover, once the assignment files are sent to the Remote Server for the analysis of index of similarity, the plagiarism-detection algorithm will only concern itself with the following file types:

1. temp.java
2. temp.c
3. temp.cpp
4. temp.h
5. temp.hxx

## Software Interfaces

### 4.3.1 Login Authentication

The system will compare the user’s the login information with a database table of MOPP users to authenticate both Students and Instructors.

## Communications Interfaces

### 4.4.1 Data Transfer Protocols

The system will utilize data transfer protocols to facilitate transfer of information from the user to the Local Repository and from the Local Repository to the Remote Server. The specific protocol has not been determined however potential candidates are:

1. TCP/IP
2. HTTP
3. SSH
4. UDP

# Other Nonfunctional Requirements

## Performance Requirements

REQ-1: Processing of submissions and reporting back to the Instructor must not exceed 12

hours.

REQ-2: After the user enters his or her credentials, the MOPP authentication process shall

not exceed 15 seconds. This accounts for any connection delays with the system.

REQ-3: The user interface shall be responsive with less than 3 second wait times when

loading a new page on an internet connection of 10Mbps or faster.

REQ-4: The time between submitting an assignment and displaying a confirmation

message for the user should not exceed 5 seconds.

## Safety Requirements

REQ-1: The analysis of similarity shall use neutral wording to present results in a way that

attempts to not influence the Instructor for or against a student. For example, the results file should not say “Plagiarism detected.” Instead it should say, for instance, “90% similar.”

REQ-2: MOPP must present results in the most neutral way. For instance, when showing

results, names could be obscured. The rational is that a similarity score should be the sole reason for inspection. This does not preclude an Instructor from investigating a particular Student if they have suspicions since the option to search for a particular student will still be available.

## Security Requirements

REQ-1: No personal information shall be transmitted from the Local Repository to the

Remote Server.

REQ-2: There should be no persistent data on the Remote Server. Once the analysis results

file is returned to the Local Repository, there will be no submission data is stored in the Remote Server.

REQ-3: Students shall only have access to their own work and not the work of others.

REQ-4: Instructors shall only have access to their own current and past courses. They shall

only have access to courses from other instructors if the system administrator allows it.

REQ-5: Students and Instructors must log into the MOPP system using their assigned

MOPP account and password.

## Software Quality Attributes

REQ-1: The MOPP system shall take less than 10 minutes to learn for the Student.

REQ-2: The MOPP system shall take less than 1 hour to learn for the Instructor.

REQ-3: The MOPP system shall be available for use during university hours (i.e. 6:00 am –

11:00pm).

REQ-4: In terms of reliability, during university hours, any system failure or downtime

should not take longer than 1 hour. In the case that the MOPP system experiences any issues, the system should prevent any more uploads and reset.

REQ-5: The process of resetting the MOPP system shall not exceed 5 minutes.

REQ-6: After university hours, the MOPP system will performance maintenance checks that

should last for 15 minutes.

REQ-7: The MOPP system must be compliant with open source software (if any) licenses

and restrictions.

REQ-8: In terms of usability, the MOPP system must allow multiple Students to submit

assignments at the same time.

# Other Requirements

Appendix A: Glossary

**MOPP**: Measure of Plagiarism Percentage

**MOSS:** Measure of Software Similarity

**COSC**: Computer Science

**LR**: Local Repository

**Sandcastle**: Brock University’s Computer Science Server

**SSH**: Secure Socket Shell

**TCP**: Transmission Control Protocol

**UDP**: User Datagram Protocol

**HTTP**: Hypertext Transfer Protocol

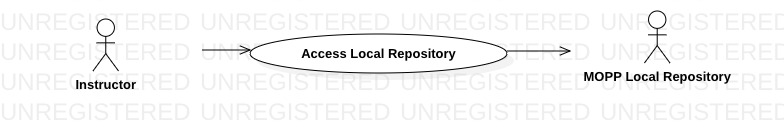
Appendix B: Analysis Models and UI Mockups

Use Case: Students submits assignment to MOPP Local Repository

****

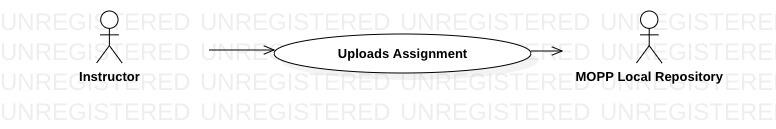
|  |  |
| --- | --- |
| **MOPP: Students submits assignment** | |
| Actors | Student, MOPP Local Repository. |
| Description | A student submits his or her assignment to the MOPP Local Repository. The data transferred includes student username, student number, and all assignment files compressed into a single .zip file. |
| Data | Student username, Student Number, Compressed (.zip) file. |
| Stimulus | Assignment submission issued by student enrolled in the course. |
| Response | Notification message and email that the assignment has been submitted correctly. |
| Comments | The student must specify what course code and assignment number during he or she is submitting to. |

Use Case: Instructor accesses MOPP Local Repository



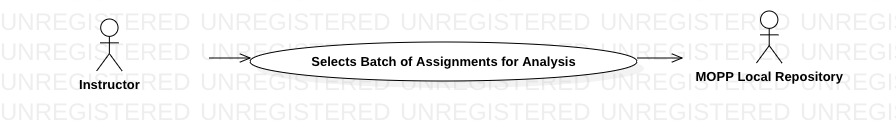
|  |  |
| --- | --- |
| **MOPP: Instructor accesses Local Repository to view students’ submissions** | |
| Actors | Instructor, MOPP Local Repository. |
| Description | An instructor may log into the MOPP Local Repository to view the current assignment submissions based on the course and assignment number. |
| Data | Students’ assignment submissions for current offering of the course. |
| Stimulus | Authentication procedure issued by instructor. |
| Response | After successful authentication, the instructor is granted visibility of assignments currently stored in the Local Repository. |
| Comments | The instructor will not have access to other instructors’ submissions unless system administrator authorizes it. Instead, the instructor can only view assignment submissions for courses that he or she currently teaches. |

Use Case: Instructor uploads assignment directly to MOPP Local Repository

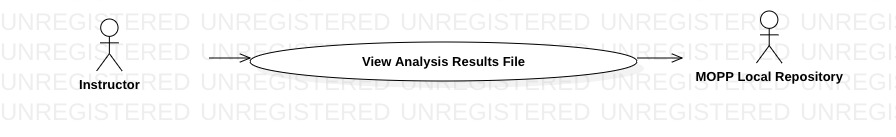


|  |  |
| --- | --- |
| **MOPP: Instructor uploads assignment to Local Repository** | |
| Actors | Instructor, MOPP Local Repository. |
| Description | An instructor may log into the MOPP Local Repository and upload an assignment to the Local Repository. |
| Data | Student Assignment. |
| Stimulus | Assignment submission issued by instructor. |
| Response | Notification message that the assignment has been submitted correctly. |
| Comments | An instructor can upload student assignments from previous offerings of the course in order to, for instance, further diversify the selection of assignments for the analysis of index of similarity. |

Use Case: Instructor selects batch of assignments in Local Repository

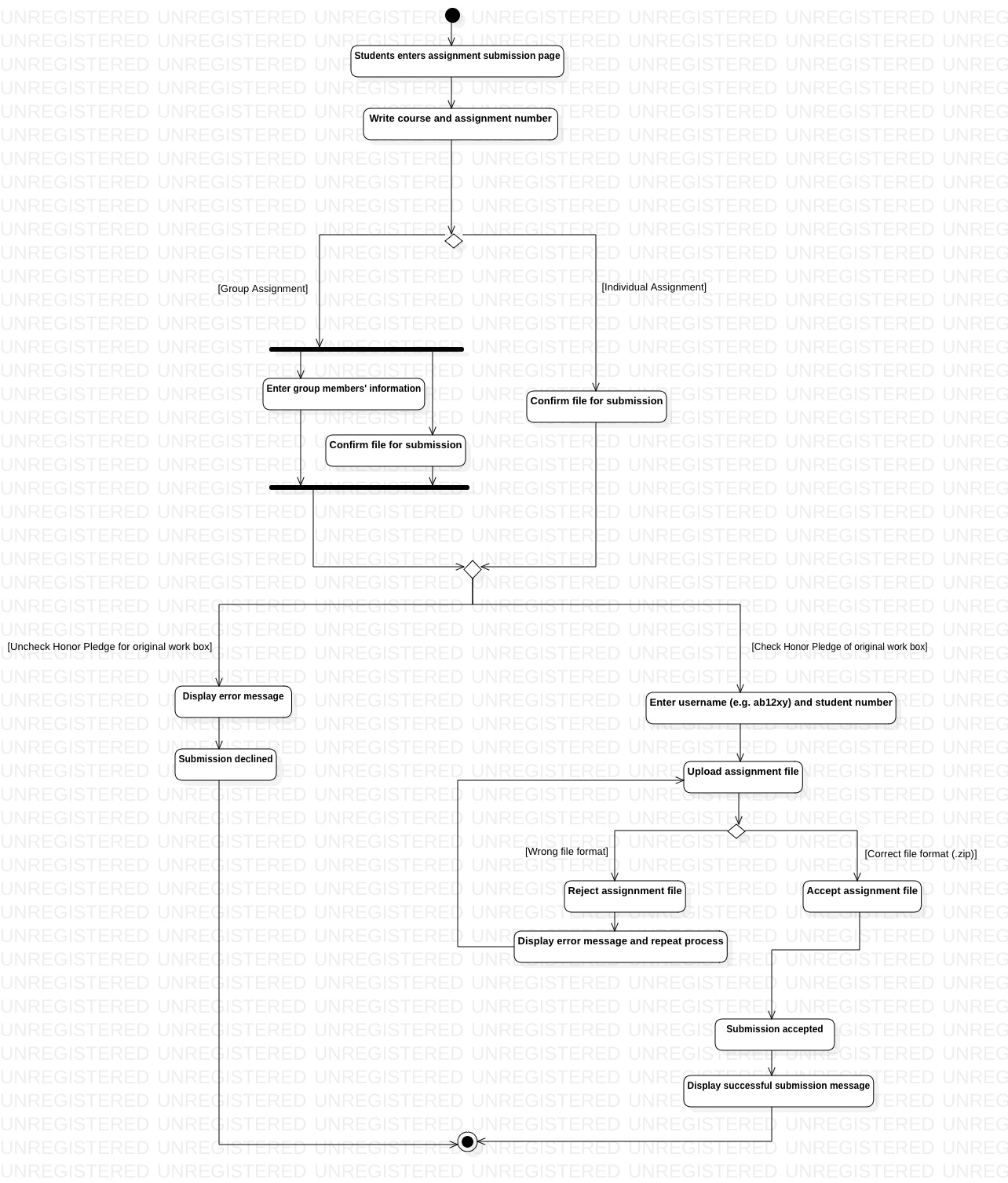


|  |  |
| --- | --- |
| **MOPP: Instructor selects batch of assignments in Local Repository** | |
| Actors | Instructor, MOPP Local Repository. |
| Description | Through the Front-End, an instructor may select a number of student assignments that need to be sent to the MOPP Remote Server for analysis. Then, the MOPP Local Repository will pre-process (i.e. removal of personal information) and tokenize the files that will be sent to the MOPP Remote Server. |
| Data | List of selected assignments from the Front-End. |
| Stimulus | Assignment Selection operation issued by instructor. |
| Response | Confirmation message that compressed file has been successfully transferred to MOPP Remote Server. |
| Comments | Once the MOPP Remote Server completes and sends back the analysis of index of similarity results file to the MOPP Local Repository, the instructor that performed the initial analysis request will be notified via email that the results are ready. |

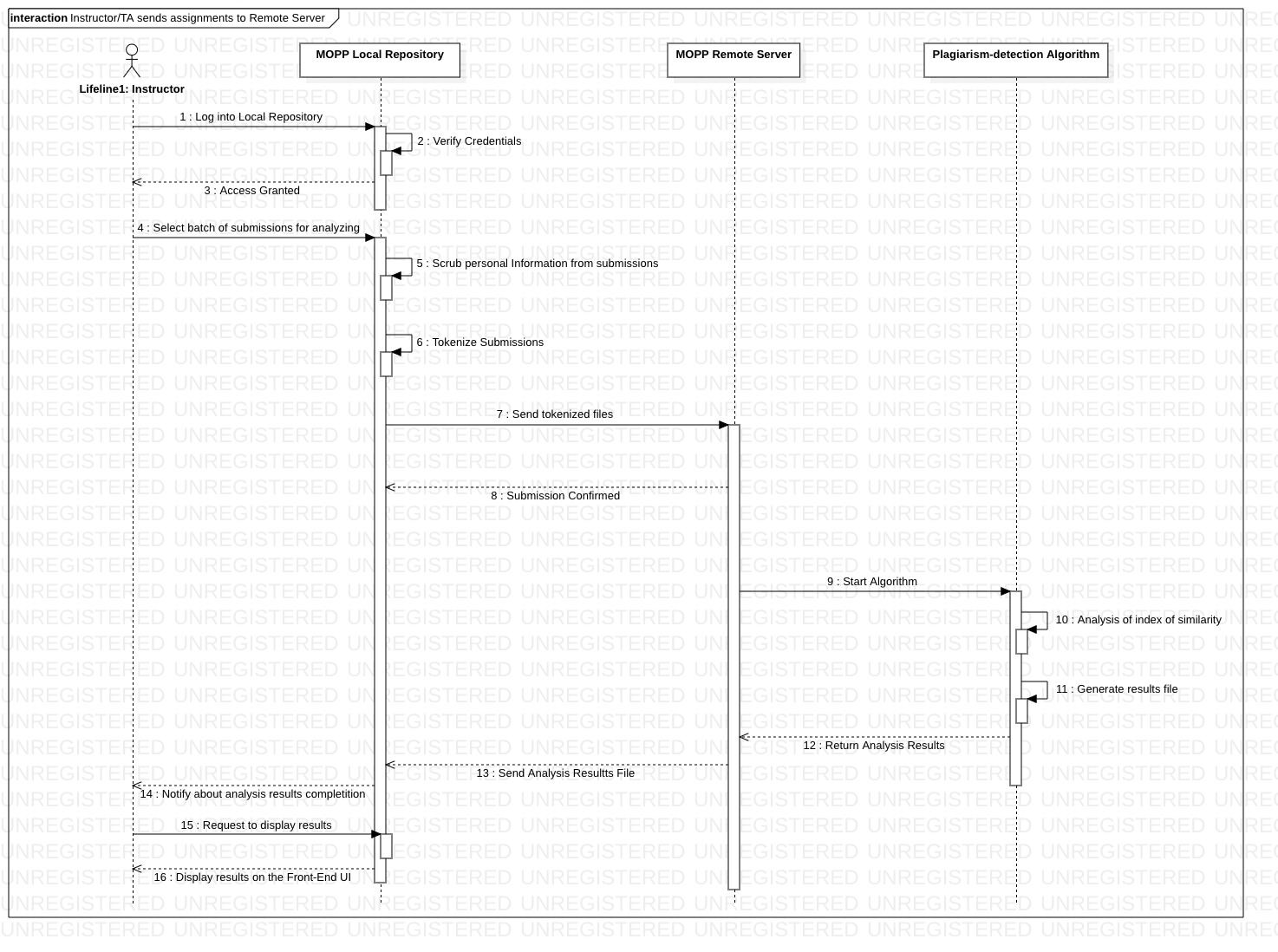
Use Case: Instructor view analysis results file in MOPP Local Repository

|  |  |
| --- | --- |
| **MOPP: Instructor view analysis results file in Local Repository** | |
| Actors | Instructor, MOPP Local Repository. |
| Description | An instructor may log into the MOPP Local Repository to view the analysis of index of similarity results file, which was received from the MOPP Remote Server after plagiarism-detection algorithm completed its work. |
| Data | Analysis of index of similarity results file. |
| Stimulus | Authentication procedure issued by Instructor. |
| Response | After successful authentication, the instructor that submitted the initial analysis request is granted permission to view the analysis results file in the MOPP Local Repository. |
| Comments | Once the MOPP Local Repository receives the analysis results file from the MOPP Remote Server, the Instructor that submitted the initial request for analysis is notified via email that the results are ready to be viewed in the MOPP Local Repository. |

Activity Diagram: Process of a student submitting an assignment to the MOPP Local Repository

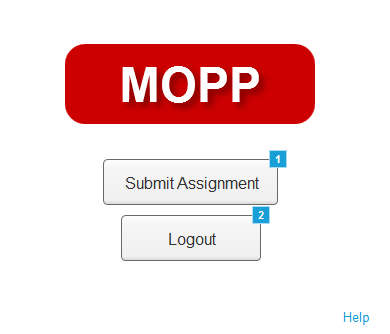


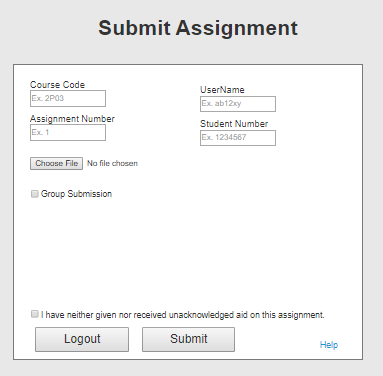
Sequence Diagram: Instructor sends assignments to MOPP Remote Server

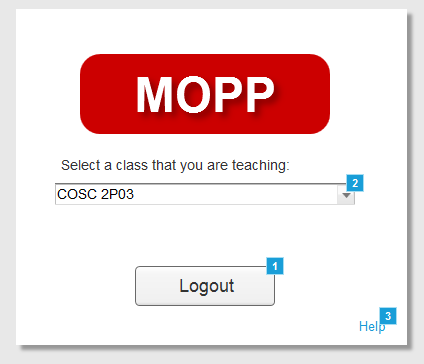


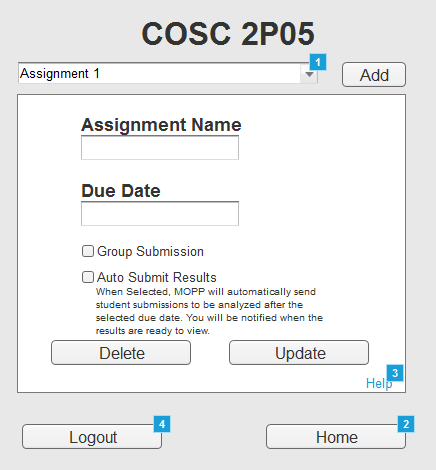
Front-End: Login Page

Front-End: Student – Home Page

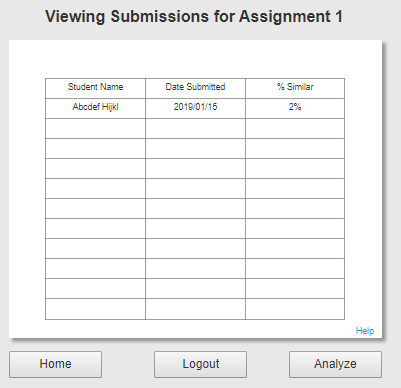


Front-End: Student – Submit Page

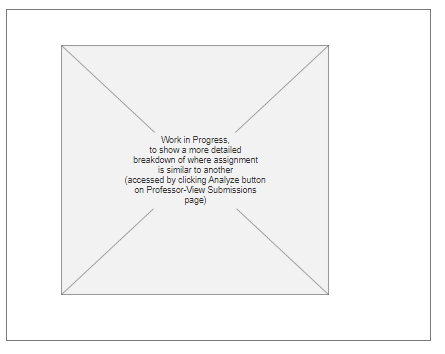
Front-End: Professor – Home Page

Front-End: Professor – Assignment Page

Front-End: Professor – View Submissions Page



Front-End: Professor – Investigate Results



Appendix C: Issues List

1. The design for displaying analysis results to the Instructor will be determined once the development team decides which information can be calculated through the algorithm.
2. The method for storing the results for transmission to the Local Repository will change based on what analysis information needs to be sent and how to display it.
3. The exact networking protocol utilized in implementation will be decided upon as the MOPP system is developed.
4. Once the project has fulfilled its primary goal, any additional features may be implemented, as outlined in this document: <https://github.com/4F00MOPP/MOPP/blob/master/Stretch%20Goals.docx>
5. Due to an unexpected bug, the Local Repository Submission Listener requires a delay during operation, as outlined in this document: <https://github.com/4F00MOPP/MOPP/blob/master/Issue%20-%20ServerStart%20Bugs.docx>