Requirements Specification

for Water Fern

Prepared by
Patrick Bright - 3860483
Goran Botic - 5638069
Melissa Locquiao - 5449384
Joey Roorda - 5629274
Cody Dennis - 5925250
Steven Thai - 5661707
Tareq Al-Masri - 5689781

4F00 Water Fern Analysis & Design

5 February 2019

Contents

Revision History

Revision	**Date**	**Author(s)**	**Description**
Added test suite to Appendix	March 23, 2019	—	Detailing the tests run
Added change log to Appendix	March 18, 2019	—	Bug as they appeared and were fixed
Removed subsubsection headings in the table of contents	March 13, 2019	—	Removes reoccurrence of subsection headings to reduce the size of the TOC
Added Chapter 7	March 13, 2019	—	Discussing implementation details
Chapter 6 edits	March 6, 2019	—	Revision and rewording of document sections

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to provide a detailed description of Water Fern, a software similarity detection system. This document will outline the purpose, objectives, and goals of the system. It will also explain the features, requirements, constraints and interfaces of the system. This document is intended for stakeholders of the system and a reference in developing the first version of the system for the development team.

1.2 Intended Audience and Reading Suggestions

This document is intended for developers, project managers, testers, documentation writers and users with a technical background attempting to understand initial goals of the project. Chapters 2 and 3 are intended as high-level descriptions of the software for users and stakeholders. Chapter 4 is intended as a guide for developers who will be implementing the system. Separate tutorials will be created for the system users. Chapter 5 is intended to provide a framework of communication for the various components of the system.

1.3 Product Scope

This document describes a system intended to be used by educational institutions for the purpose of detecting similarity in software written by students. This system allows students to submit their assignments to the school's server and the collection of works are automatically and securely processed by a remote server, before being stored back on the school's server. The purpose of this software is to help detect instances of academic dishonesty, and allows the users to easily and securely check student work. The implementation ensures that no works are saved outside of the institution's own server. This allows students to maintain their intellectual property, and allows the institution to maintain complete control over submit assignments. This ensures that the property and personal information of students is not appropriated by a third party. This software is meant to be used by students for assignment submissions, and faculty/staff for collecting assignments and comparing similarities for cases of academic dishonesty.

Chapter 2

Overall Description

2.1 Product Perspective

Water Fern is a software similarity detection software aimed at providing safe similarity analysis without compromising the information or intellectual property of students.

Background
Plagiarism is a difficult concept to define, as it can be interpreted many ways between different schools, departments, and countries. Maurer *et al.* (2006) [4] establish several criteria for their definition, including "turning in someone else's work as your own" and "changing words but copying the sentence structure of a source without giving credit". Given the ubiquity of the internet, students are able to access vast amounts of information when working on assignments. Though this resource is incredibly helpful for students doing research, this also poses the problem that educators do not know whether the work their student submits is entirely their own. In a poll from Maclean's, it was found that up to 66% of students in Canadian Universities reported cheating at least once, while up to 16% reported cheating more than 4 times [3]. The poll also showed that Computer Science was the program with the sixth highest rate of student cheating.

Students in program with higher course-loads (such as the sciences) are more likely to cheat to help offset the amount of work given to them [2]. This, coupled with myriad sources of online code such as SourceForge, github, and other online code repositories, could be major contributors to student plagiarism. According to Gallant *et al.* [2], though 55% of Japanese students self-reported cheating, only 2.8% were caught. This gap clearly shows that there is a need for methods of detecting instances of academic dishonesty.

TurnItIn.com is a commonly used solution for checking student assignments against the internet and other student submissions, however this software has limitations. Firstly, it is meant for analysis for natural language assignments (such as essays and reports). The nature of computer languages would likely not provide reliable results for said assignments. A second problem with TurnItIn.com is that many Canadian Universities and their students have become wary about sharing their work with a 3rd party company in a different country. Some Universities have even banned the use of TurnItIn.com to help protect their students' intellectual property.

Unlike TurnItIn.com, there are solutions available that are geared specifically toward finding similarities between code written by students and other sources. Currently Stanford's Measure of Software Similarity (MOSS) is used by many University Computer Science departments. Though this tool provides the desired analysis, there are still a few issues with how they implement this service. Firstly, MOSS is run on server in the United States, which brings up similar privacy concerns to that of using TurnItIn.com. Though MOSS does not claim to retain information on their server for more than 14 days [1], Universities may still have policies which do not allow them to send student information across borders. Secondly, MOSS sends a link with the results to the instructor who has submitted the assignments. This link could be intercepted and viewed by anyone. Lastly, the results of the analysis can only be viewed by the instructor for as long as they are hosted on the server, after which time they are removed from the server.

Proposed Solutions

Water Fern is an elegant, easy-to-use solution that will help to address the issues posed from sending and storing information across borders. The Water Fern system uses a front-end server installed on the client side which holds and organizes all student assignments. These files are sent via an encrypted connection to a remote server which analyzes similarities between assignments and sends and stores all results back to the front-end system. Data is only stored at the back-end during analysis, and once the analysis is complete the results are stored on the client-side. The back-end server is hosted at Brock University, in St. Catharines, Ontario, which keeps all assignments from crossing the border. In addition to this, student information is never saved on the back-end, which means that even international users can feel at ease about their information being protected.

2.2 Product Functions

Water Fern offers a platform for students to submit their assignments, and for instructors to retrieve assignments and view similarity reports. Briefly, the system will allow:

- Users to log in to the system by using their school user ID and password.
- Users to navigate an intuitive front-end interface, and easily access tutorials and FAQs for help.
- Students to submit their course assignments for similarity analysis, as well as assessment, in one step.
- Instructors and TAs to download original student submissions for assessment, unaltered by back-end processing.
- Instructors and TAs to perform similarity searches against other user data.
- Instructors to access and/or modify information and assignment information.
- Administrator to create classes and assign instructors, TAs, and students to classes.
- Administrator to add students to classes, in order to get their submission information and grant student access.
- Instructors to access similarity reports in order to inspect cases of suspected academic dishonesty.
- Instructors to collect data about assignments submitted by students, including number of assignments submitted, assignment time statistics, document length and average similarity.
- Administrators to customize similarity analysis parameters in order to return tailored similarity reports
- Effortless assignment processing by incrementally running analysis on back-end servers as students submit their assignments.
- Encrypted communication between front-end and back-end servers, ensuring that student data is kept secure.
- Assignment storage on the front-end server, ensuring that student work is only ever stored at the school in question.
- Comparison of student work from previous years' courses against current offerings' assignments.

These functions and more are discussed in greater detail in Chapter 3 System Features.

2.3 User Classes and Characteristics

There are four users that will interact with this system: student, administrator, instructor, and teaching assistant.

The average instructor should be able to use this product with little to no issue and does not require technical experience to use the system. They will also use the system the most and are the prioritized user. Instructors can add assignments to their classes for students to submit to. They can view individual student submissions and view an assignment analyses consisting of similarity scores for each individual assignment. Instructors can also submit fragments of code, which can be code provided to the students for use or reused code, for the system to disregard when finding similarities among submissions. They can also upload assignments that are found on the internet to include to find similarities, upload assignments from current students that submitted after the submission date, and can download individual assignment submissions. Instructors can also assign teaching assistants to classes and generate submission URLs to give to students to enroll in classes.

Administrators have the highest privilege level and should have a technical background and strong understanding of the system. They will use the system mainly at the start of the semester. Administrators can create and delete classes and give rights to instructors for each class. Administrators can add and remove students to classes and can also create, edit, hide and delete assignments for each class. Administrators should have higher permissions than instructors. By default, instructors should only have control over classes they own, and administrators may give instructors higher permissions later.

Teaching assistants have a lower privilege level than Instructors. The average TA will be able to do the same functions as Instructors aside from adding assignments to classes. They can view individual student submissions and view an assignment analysis that consists of similarity scores for each individual assignment. Teaching assistants can also download individual assignment submissions and upload segmented code for the system to disregard in the similarity analysis.

The average student does not require technical experience and should be able to submit their assignment or project with little to no issue. The average student will not require any new technical skills to use the system. Basic web browsing experience will be sufficient to allow a student to easily upload an assignment. Students will be using this system through out the semester. A student will select the class, select the assignment they wish to submit to, upload their assignment submission, agree to an academic pledge, and submit their assignment. Once a student submits their assignment, they will receive a confirmation that their assignment has been successfully uploaded and received. They do not see their processed assignment similarity score.

For product maintenance, a user with significant technical experience should be employed to keep the product running as intended. This user should have at least a University degree and be competent enough to know how to properly analyze and debug the product as necessary.

2.4 Operating Environment

The operating environment at the time of production uses the Brock University servers. Water Fern is designed to be relatively easy to implement into other university servers and can be dynamically adjusted for use and application. The hardware platform is a Linux-based system with sufficient computing power for completing and organizing similarity searches over time with few system failures. The only user who has access to direct server information is the administrator.

All other forms of users access a web application that handles all server or database interactions and responses. The web application is designed using common up-to-date languages, all of which should be supported by any computer built within the last fifteen years. Since the application is web-based, the computer's operating system would not be integral in the execution of Water Fern.

2.5 Design and Implementation Constraints

Restraints regarding design and implementation include the following:

- Water Fern shareholders must approve and support the choice of design for the front-end.
- The front-end of Water Fern must be designed to be accessible to all students and flexible enough to meet the requirements of different universities.
- Water Fern must be built around privacy, as information must not be taken from an unauthorized outside source.
- Water Fern must adhere to the Ontario Freedom of Information and Protection of Privacy Act (FIPPA), and Canadian Federal privacy laws.
- Water Fern should be able to be fully customized as an addition to another front end server when integrated.
- Water Fern should be able to support other major languages. At the minimum, it must support English and French (the two official languages of Canada).
- Water Fern should not have back-end processes which rely on anything that works external to Canada or Canadian servers (e.g. sensitive information does not pass the border for processing).

2.6 User Documentation

There will be an online tutorial function given for first time users to familiarize themselves on how to use this application. This tutorial function may be reactivated at any point by the user if need be. A secondary link can be accessed which redirects to a common questions page for further help and explanation, along with contact information of the administrator.

The administrator will have user manuals and implementation documentation in order to integrate the back-end portion of the system to their server or servers. They will also have user manuals to help start up the front-end and database portions of the application.

All documentation may be accessed freely by any user for more information. These documents will be able to be downloaded from the Water Fern project on Github.

2.7 Assumptions and Dependencies

Assumptions include the following:

- Front-end users are familiar with web-browsing services and can understand instructions and tutorials.
- Server is a Linux-based system with sufficient computing power to act as a web server to the user base.
- University has a Linux-based system with sufficient enough computing power for their servers.
- The client is able to authenticate user data from the front-end.

Chapter 3

System Features

This section provides high level features which users will expect from the product. This section corresponds to the user functional requirements.

3.1 Browse Student Assignment Database

3.1.1 Description and Priority

The instructor and TA are able to browse through assignments submitted to the database for analysis. This is a high priority feature since student work will only be accessible through the database.

3.1.2 Stimulus/Response Sequences

Once a course and assignment have been selected, the instructor or TA will have a visual display showing all submissions in the database for that course or assignment pair, as can be seen in A.14 and A.15. The user will be able to browse student submissions through the browser window and select one or more assignments.

3.1.3 Functional Requirements

4.1 Interface with Authentication Server, 4.4 Similarity Analysis If we expect the database browser to also display reports about each assignment, then this feature depends on the similarity analysis function, 4.8 File Encryption, 4.18 Get Option From User, 4.22 List Classes, 4.23 List Offerings, 4.24 List Assignments, 4.28 Check Permissions

3.2 Login

3.2.1 Description and Priority

The user will be able to log in to the system using their user name and password. This is medium priority since it allows the system to differentiate between users and grant them access to the system based on their permissions, but is not strictly required for the system to function.

3.2.2 Stimulus/Response Sequences

User is provided a login screen. User enters their username and password, as can be seen in A.13, A.14, A.15, and A.16, with the respected user. Upon valid username and password, the user is granted access to the system. If the user inputs incorrect credentials, they are asked to try again. 3 incorrect attempts locks the account for 10 minutes.

3.2.3 Functional Requirements

4.1 Interface with Authentication Server,A.2 General Login for all users

3.3 Student Assignment Submission

3.3.1 Description and Priority

Students will be able to select the course and assignment to submit, and then upload their files through the system. This is low priority since the batch upload feature for the instructor is sufficient to have files uploaded. This feature will help to offload the work needed on the instructor's end.

3.3.2 Stimulus/Response Sequences

Once the student has selected their course/assignment, they are presented with a screen that allows them to upload files from their computer, as can be seen in A.16. The user clicks the upload button and selects the files to be uploaded. There are customizeable checkboxes they must tick (e.g. honour pledge) in order to upload files. When the student clicks upload, the files are sent to the front-end server database. The student receives an e-mail notification when their files are successfully sent to the system.

3.3.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.4 Similarity Analysis, 4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.18 Get Option From User, 4.5 Document Indexing, 4.20 Store Assignments, 4.28 Check Permissions,A.5 Student Submission

3.4 Bulk Assignment Submission

3.4.1 Description and Priority

It will be possible to submit a batch of assignments to a course in an automated manner. This is considered to be a high priority feature since it is assumed that instructors managing a course are more likely to use existing submission systems, rather than the built in submission system. Furthermore, a single assignment submission system can be implemented using the batch submission system.

3.4.2 Stimulus/Response Sequences

The instructor's course assignment page will have an upload page which will allow the instructor to upload multiple assignments all at once, as can be seen in A.14 There will be a choice to either drag/drop all assignments to the page or select the files/folders they want to upload.

Assignment upload will occur as soon as the files and/or folders are dropped on the page or selected via the 'Upload Files' button.

3.4.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.4 Similarity Analysis, 4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.18 Get Option From User, 4.5 Document Indexing, 4.20 Store Assignments, 4.28 Check Permissions

3.5 Download Files

3.5.1 Description and Priority

Once an assignment has been uploaded to the database, Instructors and TAs will be able to download submissions. This is high priority since this is the main method by which the assignments can be retrieved. Most computer science assignments need to be run for marking, so downloading the raw files is incredibly important.

3.5.2 Stimulus/Response Sequences

Once a class or assignment has been selected, the user is presented with all submitted assignments, which can be seen in A.14 and A.15. The user can select individual assignments, or choose to download all assignments. Once files have been selected, the user clicks download and files are downloaded at a specified directory.

3.5.3 Functional Requirements

4.1 Interface with Authentication Server, 4.3 Download Files, 4.8 File Encryption, 4.18 Get Option From User, 4.28 Check Permissions

3.6 Web Tutorials

3.6.1 Description and Priority

Although the system aims to be as intuitive as possible, there will also be a website tutorial available for users to get better understanding on how to use the system. This is low priority since it is not critical to the system working. There are also in-depth pages about how to use the system tools.

3.6.2 Stimulus/Response Sequences

The standard display window will have a button that presents the user with a visual display that guides the user through all the features on their screen. The user clicks next to go to the next feature. Each feature will have a blurb on what it does and how to use it, as well as a link to a more in-depth explanation on how to use the tool.

3.7 Similarity Report Generation

3.7.1 Description and Priority

This system feature will be the main focus of the software application that aggregates student assignment submissions for similarity evaluation. This is not to flag assignments for plagiarism as this will be the responsibility for the Instructor or TA given the statistics provided in this feature. This feature is of the highest priority since it is the main component of the system.

3.7.2 Stimulus/Response Sequences

A new window will contain two sub windows for the student assignment submission and the most similar assignment stored in the database. There will be a progress bar that will indicate the percentage of similarities between assignments. The Instructor or TA will have access to specialized data analysis tools to inquire further analysis. Once evaluation is completed and all assignments are processed, the Instructor or TA can generate a report on all the assignments submitted that can be downloaded locally, which can be seen in A.14 and A.15.

3.7.3 Functional Requirements

4.1 Interface with Authentication Server, 4.3 Download Files, 4.4 Similarity Analysis, 4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.17 CSV From SQL Table, 4.18 Get Option From User, 4.27 Select Code Block, 4.28 Check Permissions,A.8 Similarity Checking .

3.8 Create and Modify Classes

3.8.1 Description and Priority

By default, this feature will only be accessible to the System Administrator. The details of this feature include class creation and modification. Creating classes will have general properties and description of a class. An offering can be created from the class if it becomes active for the term. This will make modifying offerings of the same class much more efficient. This a high priority since there needs to be classes to create offerings.

3.8.2 Stimulus/Response Sequences

For creating a new class the system administrator will have a 'create' button which will display a new window containing a simple registration form that will have mandatory fields to be filled, which can be seen A.13. Once completed the offering will be registered into the database. For modifying a class the system administrator will have a 'modify' button that will be visible when an existing offering has been selected. The registration for will be displayed with the fields automatically filled from it's creation. There will also be an option for bulk class creation (via CSV).

3.8.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.11 Create Class, 4.12 Delete Class, 4.18 Get Option From User, 4.22 List Classes, 4.28 Check Permissions

3.9 Create and Modify Offerings

3.9.1 Description and Priority

Specific instances of configured classes can be created, containing their own set of students and assignments. These instances represent a single term/semester of that class being offered. Each offering will also be tied to the date that it was started as a means of book-keeping and organization. This is considered as a high priority feature because the organization of submissions by offering and assignment number will be valuable when deciding which past submissions an assignment should be compared with.

3.9.2 Stimulus/Response Sequences

The System Administrator will select an existing class, which can be seen in A.13. In the class directory it will contain all offerings of the class. There will be a button 'create offering' that will display two date fields, Beginning term and end term. Once completed offering will be available to the Instructor and TA. The offering will be available to students at the beginning of the term.

3.9.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.13 Create Offering, 4.14 Delete Offering, 4.18 Get Option From User, 4.23 List Offerings, 4.28 Check Permissions

3.10 Create and Modify Assignments Within An Offering

3.10.1 Description and Priority

Each offering may require a number of assignments to be submitted by students for the purpose of evaluation. By default the assignments associated with an offering can be inherited from the base instance of a class. However, it will be possible to configure assignments and due dates within specific offerings independent of the base class. Each assignment will have an associated due date, though it will remain possible to submit an assignment after the provided due dates. An assignment submit after the due date will be automatically flagged. This is considered as a medium priority feature because the organization of submissions by offering and assignment number will be valuable when deciding which past submissions an assignment should be compared with.

3.10.2 Stimulus/Response Sequences

The Instructor can modify an assignment's properties by selecting an assignment in the offering's assignments directory and clicking on the 'modify' button, which can be seen in A.14. A new window will be prompt with automatically filled fields written previously on creation. Name, due dates and files can be added, modified or deleted. Once completed the Instructor can submit changes with a 'submit' button and will be prompted to confirm changes.

3.10.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.15 Create Assignment, 4.21 Delete Assignment, 4.18 Get Option From User, 4.24 List Assignments, 4.28 Check Permissions

3.11 Submission Override

3.11.1 Description and Priority

Within a specific offering it will be possible to de-flag late assignments, at the discretion of the individual responsible for the offering. This feature is considered to be low priority since it will be possible to manually record which students were given permission to submit their assignments late.

3.11.2 Stimulus/Response Sequences

The Instructor will be able to filter assignments by late submission, select an assignment and click on the 'Submission Override' button.

3.11.3 Functional Requirements

4.1 Interface with Authentication Server, 4.18 Get Option From User, 4.28 Check Permissions

3.12 Front End Accessibility

3.12.1 Description and Priority

The user interfaces of the system will meet the guidelines and requirements specified by the Accessibility for Ontarians with Disabilities Act(AODA) Part II Information and Communication Standards, Section 14 Accessible websites and web content.

- https://www.aoda.ca/integrated/#awawc

In accordance with the specified section the interfaces of this sytem will conform with the WCAG 2.0 Level AA standard, published by the World Wide Web Consortium (W3C)

- https://www.w3.org/TR/WCAG20/

This will be considered as a high priority feature since the university will be responsible for providing accommodations in the event that a student is unable to use the submission system. By meeting the requirements specified by the AODA we will have ensured that all reasonable measures have been taken to provide access to the system to all students.

3.13 Enroll Students In An Offering

3.13.1 Description and Priority

Once an offering of a course has been created prior to the start of the term, the administrator will be able to add access permissions for all students enrolled in the course. This will allow students to navigate to the submission page for each offering they are enrolled in.

3.13.2 Stimulus/Response Sequences

The Instructor or System Administrator will be presented with an input box to enter the student number of the student that he/she wants to add to the course, as the instructor is typing the students name, the input box will start giving the instructor a list of student numbers that match the currently entered numbers. When the instructor hits enter, another input box will be given to enter another student number. Under all the input boxes will be a button to submit the students to the Administrator to add them to the system. Additionally, bulk enrollment via CSV files would also be an option. This can be seen in A.13 and A.14.

3.13.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.18 Get Option From User, 4.22 List Classes, 4.23 List Offerings, 4.28 Check Permissions

3.14 Manual Document Flagging

3.14.1 Description and Priority

It will be possible for the instructor and teaching assistants of a course to manually flag assignments which they consider to be suspicious. This will allow suspicious assignments to be easily tracked and accessed without requiring the teaching assistant or instructor to search through all submissions. This feature is considered to be high priority since our system is not responsible for identifying plagiarism. Our system simply identifies similarities between documents, and supports the decision making process of its users.

3.14.2 Stimulus/Response Sequences

In the evaluation phase if the Instructor or TA suspects plagiarism in an assignment on the document viewer there will be a button with a picture of resembling of a flag will notify the system and display a flag on the top right corner of the student assignment. The flag can be toggled off if the Instructor or TA made an error.

3.14.3 Functional Requirements

4.1 Interface with Authentication Server, 4.8 File Encryption, 4.18 Get Option From User, 4.28 Check Permissions, 3.1 Browse Student Assignment Database

3.15 Submission Statistics Generation

3.15.1 Description and Priority

The system will be capable of providing a number of data points regarding assignment submission to users such as teaching assistants and instructors. Some example measurements include: the number of late submissions, the time of each submissions, or the total number of submissions. Such measurements are expected to be of interest to those responsible for an offering as they may be used to measure the performance and level of engagement of students. Any measurements will be provided in a standard format such as CSV or TSV. This feature is considered to be low priority since it is not essential to the systems ability to function, and is not essential to the users of the system.

3.15.2 Stimulus/Response Sequences

The instructor user will have the permission to generate statistics. The functionality will be accessible after the assignments have been evaluated. In the evaluation window there will be a 'Statistics' button that will write a new file in a CSV or TSV which will contain the number of late submissions, the time of each submissions, or the total number of submissions. This can be seen in A.14.

3.15.3 Functional Requirements

4.1 Interface with Authentication Server, 4.3 Download Files, 4.17 CSV From SQL Table, 4.18 Get Option From User, 4.28 Check Permissions

3.16 Create and Modify Users

3.16.1 Description and Priority

It will be possible to create and modify profiles for the users of the system. While it is expected that the university's authentication server will be used to authenticate the users of the system, it will be necessary to record some information regarding the system's users. A user profile will be primarily used to record the access levels and permissions granted to users of the system. This is considered to be a high priority feature because the management of access levels and permissions is essential to the protection of system users personal information and intellectual property.

3.16.2 Stimulus/Response Sequences

The system administrator will be presented with a list of users that are currently registered to the system, there will be a search bar to make it easier to search for a currently registered user and update their profile, which can be seen in A.13 There will also be a button to add a new user to the system.

If the user hasn't been registered yet, the administrator will be presented with an 'Add New User' button. The administrator will be taken to a page where they can fill all the necessary information about the user being added.

If the user has already been registered, the administrator will be able to either select that user from the list of people on the page, or type the user's name/student number in the search box to get the user, the search box will give a drop down of users that have the same name/student number currently being entered in the search box. The administrator will be able to click on the user's name/student number as its being shown on the drop down and the website will automatically take the administrator to the users profile.

When the administrator enters the selected user's profile they will be presented with all the user's information already filled out. The administrator will be able to make the changes necessary, and when they are done updating the page will present a prompt asking them if they are sure of the changes they want to make. The administrator can either select 'No' which will take them back to the editing page, or select 'Yes' which will update the users profile and send the administrator back to the users list. There will also be a 'Confirm Changes' prompt for the administrator when saving changes.

The page will also have a 'Cancel' button so that the administrator can go back to the users page and not make any changes to the users profile.

3.16.3 Functional Requirements

4.1 Interface with Authentication Server, 4.2 Upload Files, 4.9 Create User, 4.10 Delete User, 4.18 Get Option From User, 4.28 Check Permissions

3.17 Generate Usage Data

3.17.1 Description and Priority

A user with sufficient permissions should be able to generate or poll a variety of relevant usage statistics. Some examples include generating a list of students, and the classes they're enrolled in, generating a list of the classes which have used an assignment, generating a list of offerings of a particular class, etc. This feature is considered to be low priority since it is not essential to the systems ability to function, and is not essential to the users of the system.

3.17.2 Stimulus/Response Sequences

A user with sufficient permissions will be given a filter bar where they will be able to choose what kind of information they want to see. The first options box will ask the user to select what information section they want to get. Every selection box will give the next selection box, so initially there will be one selection box to choose what kind of information you want to get from the system, when the user selects their option, the page will give the user another selection box to give sub-options for the option selected.

After all the filters have been added, the page will generate the list of information that the user wants to see.

3.17.3 Functional Requirements

4.1 Interface with Authentication Server, 4.3 Download Files, 4.8 File Encryption, 4.17 CSV From SQL Table, 4.18 Get Option From User, 4.22 List Classes, 4.23 List Offerings, 4.24 List Assignments, 4.28 Check Permissions

Chapter 4

Functional Requirements

This section describes the functionality which must be present in order to provide the features described in Chapter 3 System Features. The level of detail provided for entries in this section may vary. Wherever possible we will provide a detailed description of the required function, however certain details will, out of necessity, be implementation dependent. When this is the case to be decided (*TBD*) will be used as a placeholder.

4.1 Interface with Authentication Server

4.1.1 Description of Function

Many universities have their own authentication systems which students and/or staff use to log in to their accounts, e-mail, school computers, etc. This feature allows users to log in using their school account. This feature is low priority since it isn't necessary that students use their school accounts, custom created accounts can be used for submission. The user enters their full student account (e.g. ab19cd@brocku.ca) and the password for that account. The user's university's authentication verifies that user's credentials and allows or denies access to the system.

4.1.2 Depends On

4.1.3 Expected Inputs

A user name and password.

4.1.4 Expected Outputs

A token of some kind indicating the success or failure of authentication. The exact nature of this token is dependent on the university's system, and is unknown at this time.

4.1.5 Expected Side Effects

None.

4.2 Upload Files

4.2.1 Description of Function

This function will be used when a user attempts to copy files from their machine to the front-end server of the system. It is assumed that any data uploaded to the server may be stored on disk. To ensure the integrity of the data and the security of the system, any data saved on disk will be encrypted, thus the dependency on file encryption. The dependency on 4.28 Check Permissions exists because the ability to store data on the server will be restricted, thus it will be necessary to determine whether a user is permitted to upload data when the request to upload is made.

4.2.2 Depends On

4.8 File Encryption, 4.28 Check Permissions.

4.2.3 Expected Inputs

A location on the user's local machine.

4.2.4 Expected Outputs

An indication of success or failure.

4.2.5 Expected Side Effects

If the upload is successful a file will be stored on disk on our front end server.

4.3 Download Files

4.3.1 Description of Function

This function will be used when a user attempts to download submissions, usage data, or any other file from the server. Any data stored on disk will be encrypted, thus the dependency on file decryption. Access to data stored on the server will be restricted, and it will be necessary to determine whether or not a user is permitted to download data when the request is made, thus the dependency on 4.28 Check Permissions.

4.3.2 Depends On

4.8 File Encryption, 4.28 Check Permissions.

4.3.3 Expected Inputs

A location on the front-end servers storage device.

4.3.4 Expected Outputs

An indication of success or failure.

4.3.5 Expected Side Effects

None.

4.4 Similarity Analysis

4.4.1 Description of Function

This is the core system implemented on the back-end server, taking in the assignment data from the front-end database and performing analysis for similarity. The results are saved to the front-end database. Each time an assignment is submitted, this function will be called and the current submission will be compared against those stored in the database. The similarity reports associated with assignments currently in the database will be updated as needed, and a similarity report will be generated for the provided assignment based on the submissions currently in the database. This function depends on 4.6 Document Decomposition and 4.5 Document Indexing because document decomposition and indexing are integral to the process of similarity analysis. The dependency on 4.7 Authentication Between Front End and Back End is due to the fact that similarity analysis will involve communication between the front-end and back-end server. The back-end server is expected to read from and potentially write to the front-end database during analysis, thus regular checks must be performed to ensure that a third party is not involved in the communication. Since the back-end server is expected to retrieve indexed sub-sections of documents and may write new data to the database, 4.8 File Encryption, 4.27 Select Code Block and 4.28 Check Permissions are required.

4.4.2 Depends On

4.6 Document Decomposition, 4.5 Document Indexing, 4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.27 Select Code Block, 4.28 Check Permissions.

4.4.3 Expected Inputs

A raw assignment

4.4.4 Expected Outputs

A similarity report comparing the submit assignment against the assignments currently stored in the database.

4.4.5 Expected Side Effects

Existing similarity reports within the database will be updated as appropriate with respect to the newly submit assignment.

4.5 Document Indexing

4.5.1 Description of Function

This function refers to the ability to take a document, and produce and store references to the document or specific sub-sections of the document. This sub-document level indexing can be used to increase efficiency when comparing large documents, and to increase efficiency when retrieving similar sections of code to display to a user. Document indexing will also have some notion of versioning with respect to changes to the indexing algorithm. Changes to the indexing algorithm should be backwards compatible with older versions (within reason) and changes should be made to 4.4 Similarity Analysis to maintain this if necessary. This function will depend on 4.8 File Encryption because it may be responsible for storing processed data on disk, and data stored on disk will be encrypted before it is stored. The function 4.26 Store Code Block will be responsible for associating a sub-section of a document with an identifier, thus will require 4.26 Store Code Block. Finally any entity using this function must have sufficient permission to write data to the database, thus the dependency on 4.28 Check Permissions.

4.5.2 Depends On

4.8 File Encryption, 4.26 Store Code Block, 4.28 Check Permissions

4.5.3 Expected Inputs

A document to index.

4.5.4 Expected Outputs

TBD: This functionality will be further decomposed into simpler steps during implementation. The expected output will depend on implementation specific details.

4.5.5 Expected Side Effects

TBD: This functionality will be further decomposed into simpler steps during implementation. The expected side effects will depend on implementation specific details.

4.6 Document Decomposition

4.6.1 Description of Function

This function refers to the ability to take a document and decompose it into a number of logically separate, smaller documents. As a simple example, a class in Java may be decomposed into a number of separate constructors and methods. This functionality will be required to gain control over the granularity of comparison, and allow the efficient indexing or retrieval of only the relevant sections of a document. This functionality will depend on 4.8 File Encryption because a complete document stored on disk will be encrypted. When the document is retrieved from disk decryption will be necessary.

4.6.2 Depends On

4.8 File Encryption

4.6.3 Expected Inputs

A document to decompose.

4.6.4 Expected Outputs

TBD: This functionality will be further decomposed into simpler steps during implementation. The expected output will depend on implementation specific details.

4.6.5 Expected Side Effects

TBD: This functionality will be further decomposed into simpler steps during implementation. The expected side effects will depend on implementation specific details.

4.7 Authentication Between Front End and Back End

4.7.1 Description of Function

This function will be used during communication between the front-end and back-end servers. It will be responsible for confirming the identity of an entity claiming to be the front-end server or the back-end server.

4.7.2 Depends On

TBD: Depends on, expected inputs, expected outputs, and expected side effects will be depended on the protocol which we select for authentication.

4.7.3 Expected Inputs

4.7.4 Expected Outputs

4.7.5 Expected Side Effects

4.8 File Encryption

4.8.1 Description of Function

This function refers to the ability to encrypt and decrypt data. This function is necessary for ensuring that data in long term storage is only readable when it is necessary for it to be read. Standard AES encryption will be used. This function depends on 4.28 Check Permissions because the ability to encrypt or decrypt data is tightly coupled with system security. Suppose an adversary has gained the ability to encrypt data, but has not gained access to the encryption key. Large volumes of cipher text with known plain text could easily be generated for later analysis. Thus a permission check is required before allowing encryption or decryption.

4.8.2 Depends On

4.28 Check Permissions.

4.8.3 Expected Inputs

Data to encrypt.

4.8.4 Expected Outputs

Encrypted data.

4.8.5 Expected Side Effects

TBD: The expected side effects of this method will depend on our selected protocol for managing encryption keys.

4.9 Create User

4.9.1 Description of Function

This function encompasses the creation of user accounts. There are 4 different types of user accounts, each coming with a different set of user permissions. The default account types in decreasing level of permissions are: Administrator, Instructor, Teaching Assistant, and Student. Customized permissions may also be set so that it can deviate from these defaults. User accounts will have a username, password, and a Name associated with them, along with additional optional personal information. All account information will be stored on the front-end database, and any personal or sensitive information therein will have the highest level of database security placed on it.

4.9.2 Depends On

4.8 File Encryption User data may need to be encrypted, 4.28 Check Permissions.

4.9.3 Expected Inputs

Account type, custom permissions, username, password, name, additional user information.

4.9.4 Expected Outputs

Account ID

4.9.5 Expected Side Effects

Creation of an account object within the database

4.10 Delete User

4.10.1 Description of Function

The function refers to the deletion of user accounts. When a user account is deleted any personal or identifying information is removed from the database, however a user account stub containing the user id, and auto-generated stand-in required information like username and name will be left so that anything referring to the account can continue to function as normal.

4.10.2 Depends On

4.8 File Encryption Cannot delete encrypted data if you cannot encrypt it, 4.28 Check Permissions.

4.10.3 Expected Inputs

Username or User ID.

4.10.4 Expected Outputs

Success indicator.

4.10.5 Expected Side Effects

Replacement of the specified user account object with a stub account and removal of all user identifying information from the database.

4.11 Create Class

4.11.1 Description of Function

This function refers to the creation of new class objects. Class objects will contain a name and an optional course description. These class objects serve as an organizational header or tag to manage offering history of the course it represents. As such, the class objects may also contain metadata relevant to new or past offerings of the course. Class objects will be stored in the front-end database.

4.11.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.11.3 Expected Inputs

The expected input to this function is a class identifier. Optionally, metadata such as a course and assignment descriptions can be provided.

4.11.4 Expected Outputs

A token indicating the success or failure of the operation.

4.11.5 Expected Side Effects

The database will be updated to reflect the existence of the new class.

4.12 Delete Class

4.12.1 Description of Function

This function refers to the deletion of class objects. Deleting class objects will also remove the offering history for said class, and any assignment submissions therein, and as such, should be treated with a high level of security.

4.12.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions, 4.14 Delete Offering, 4.21 Delete Assignment.

4.12.3 Expected Inputs

The class to delete.

4.12.4 Expected Outputs

A token indicating the success or failure of the operation.

4.12.5 Expected Side Effects

The database will be purged of the class and all associated data.

4.13 Create Offering

4.13.1 Description of Function

This function is used to create an offering of a class. An offering is an organizational unit representing a period for which a class was offered. An offering is used to reference assignments submitted by students taking a class during a specific period, thus when a student or instructor submits files for similarity analysis, they submit the files not to a class, but to an offering. An offering is created from a classes, when created the offering will inherit metadata, such as a course description or assignments, from a class.

4.13.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.13.3 Expected Inputs

The offering identifier, and the class to inherit from.

4.13.4 Expected Outputs

A token indicating the success or failure of the function.

4.13.5 Expected Side Effects

The database will be updated to reflect a new offering.

4.14 Delete Offering

4.14.1 Description of Function

This function allows an offering to be deleted from the database. The deletion of an offering will also cause all assignment submissions of that offering to be deleted from the database, as a result, this action should be treated with a high level of security. The assignments submitted to an offering may have been indexed by the back-end sever. If this is the case, references to the assignments associated with the offering should be purged from the database. This may involve updating similarity reports, and should leave the database in a consistent state without dangling references.

4.14.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.14.3 Expected Inputs

The identifier of the offering to delete.

4.14.4 Expected Outputs

A token indicating the success or failure of the operation.

4.14.5 Expected Side Effects

The database will be updated as needed so that the offering and all associated data is no longer present.

4.15 Create Assignment

4.15.1 Description of Function

This function allows an assignment to be added to a class or offering. An assignment serves as an organizational unit for managing submissions. Once an assignment is added to a class, all subsequent offerings of the class will inherit the assignment, although previous offerings of the class will not be updated to have the assignment. If an assignment is added to an offering only that specific offering will be effected.

4.15.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions

4.15.3 Expected Inputs

An assignment identifier, optionally a description of the assignment can be provided, and a class or offering to associated the assignment with.

4.15.4 Expected Outputs

A token indicating whether or not the assignment has been successfully added to the class or offering.

4.15.5 Expected Side Effects

The provided class or offering will be modified to contain the assignment.

4.16 Remove Assignment

4.16.1 Description of Function

This function is used to remove an assignment from a class or offering. An assignment serves as an organizational unit for managing submissions. If an assignment is removed from a class, all subsequent offering will no longer inherit the assignment, however previous offerings of the class will not be effected. If an assignment is removed from an offering only that specific offering will be effected and all submissions of that assignment will be removed from the database. If an assignment was indexed by the backend server, then all references to the assignment should be purged from the database. This may involve updating similarity reports, and should leave the database in a consistent state without dangling references.

4.16.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions

4.16.3 Expected Inputs

An assignment identifier and a class or offering identifier.

4.16.4 Expected Outputs

A token indicating whether or not the assignment was successfully removed.

4.16.5 Expected Side Effects

The provided class or offering will be modified to no longer contain the assignment.

4.17 CSV From SQL Table

4.17.1 Description of Function

This function is responsible for taking the result of a select statement, and converting it to CSV format. This will be used when preparing data for users of the system to export.

4.17.2 Depends On

4.8 File Encryption Some of the data used to generate CSVs may be encrypted.

4.17.3 Expected Inputs

The result of a select statement run on the database.

4.17.4 Expected Outputs

A CSV file representing the passed in data.

4.17.5 Expected Side Effects

None.

4.18 Get Option From User

4.18.1 Description of Function

This is a generalized function which can be used to present fields to the user, and retrieve user input. Default values, acceptable ranges, etc, for fields should be passable as arguments to this function. The function should inform users when the values they provide are invalid, and provide advice on how to correct the issue when possible. The function should allow users to replace invalid values.

4.18.2 Depends On

4.19 Validity Checker

4.18.3 Expected Inputs

The expected input to this function is a list of pairs (value type, validity checker) where value type describes the type of the expected value, and validity checker is a function which determines whether or not a provided value is valid. Validity checker should return a token which either indicates that the value is valid, or describes why the value is invalid.

4.18.4 Expected Outputs

The set of values provided by the user.

4.18.5 Expected Side Effects

None.

4.19 Validity Checker

4.19.1 Description of Function

This describes a class of functions used by 4.18 Get Option From User. A validity checker is responsible for determining if a value provided by the user is valid. A number of validity checkers must be defined for the various types of input which must be retrieved from the user.

4.19.2 Depends On

4.19.3 Expected Inputs

A value provided by a user.

4.19.4 Expected Outputs

A token indicating either that the provided value is valid, or why the value is invalid.

4.19.5 Expected Side Effects

None.

4.20 Store Assignments

4.20.1 Description of Function

This function will store a raw assignment in the database. The assignment will be indexed according to offering and assignment.

4.20.2 Depends On

4.8 File Encryption, 4.28 Check Permissions

4.20.3 Expected Inputs

An offering, assignment ID and the raw data of the assignment.

4.20.4 Expected Outputs

A token indicating whether or not the assignment was successfully stored in the database.

4.20.5 Expected Side Effects

Function is expected to update the database as appropriate.

4.21 Delete Assignment

4.21.1 Description of Function

This function is used to delete an assignment from the database. If the assignment has been processed then there should exist a number of references to various pieces of code in the assignment. This function should purge the database of all references to code contained within the specified assignment.

4.21.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.21.3 Expected Inputs

The assignment to delete.

4.21.4 Expected Outputs

A token indicating whether or not the assignment was successfully deleted

4.21.5 Expected Side Effects

This function is expected to remove the assignment, and all references to it from the database.

4.22 List Classes

4.22.1 Description of Function

This function should be capable of producing a list of the classes currently represented in the database. It should be possible to tailor the results included in the list through arguments to the function.

4.22.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.22.3 Expected Inputs

A collections of flags describing the desired classes. Some examples include a set of course codes, a department or faculty name, or keywords.

4.22.4 Expected Outputs

A collection of classes.

4.22.5 Expected Side Effects

None.

4.23 List Offerings

4.23.1 Description of Function

This function should be capable of producing a list of the offerings currently represented in the database. It should be possible to tailor the results included in the list through arguments to the function.

4.23.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.23.3 Expected Inputs

A set of flags describing the desired offerings. Some examples include the class which the offerings should represent, the instructor which taught the desired offerings, or students for which you would like to see the classes that they have taken.

4.23.4 Expected Outputs

A collection of offerings.

4.23.5 Expected Side Effects

None.

4.24 List Assignments

4.24.1 Description of Function

This function should be capable of producing a list of the assignments currently represented in the database. It should be possible to tailor the results included in the list through arguments to the function.

4.24.2 Depends On

4.8 File Encryption same rational as user data, 4.28 Check Permissions.

4.24.3 Expected Inputs

A collection of flags describing the desired assignments. Some examples include classes from which to take assignments, students whose assignments you would like to view, etc.

4.24.4 Expected Outputs

A collection of assignments.

4.24.5 Expected Side Effects

None.

4.25 Select Assignment

4.25.1 Description of Function

This function should allow retrieval of a collection of assignments according to the offerings and assignments. This function should also support wild cards and ranges as elements of the offerings and assignments.

4.25.2 Depends On

4.8 File Encryption, 4.28 Check Permissions

4.25.3 Expected Inputs

A offerings and assignments describing the desired assignments.

4.25.4 Expected Outputs

A collection of assignments matching the provided offerings and assignments.

4.25.5 Expected Side Effects

None.

4.26 Store Code Block

4.26.1 Description of Function

This function should allow a specific section of code to be associated with an index. Both the block of code and the desired index should be provided as arguments to the function. The provided index values should be reflective of the content of the code, this will allow similar sections of code to be retrieved from the database without the need to examine entire documents.

4.26.2 Depends On

4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.28 Check Permissions

4.26.3 Expected Inputs

The expected inputs of this function are a reference to the section of code, and an index value used to identify it.

4.26.4 Expected Outputs

The expected output of this function is a token indicating whether or not the section of code was successfully associated with the provided index.

4.26.5 Expected Side Effects

This function will updated the database to associate the provided section of code with the provided index.

4.27 Select Code Block

4.27.1 Description of Function

This function allows the retrieval of a block of code according to a provided index. It should be possible to specify a range of values for the index. The index value is reflective of the content of the code, thus this function allows for the retrieval of a collection of similar pieces of code without the need to examine entire documents.

4.27.2 Depends On

4.7 Authentication Between Front End and Back End, 4.8 File Encryption, 4.28 Check Permissions

4.27.3 Expected Inputs

The expected input of this function is a range of index values describing the desired code segments.

4.27.4 Expected Outputs

The expected output of this function is a collection of index values which correspond to code stored in the database and fall within the provided range.

4.27.5 Expected Side Effects

None.

4.28 Check Permissions

4.28.1 Description of Function

This function defines the ability to check access rights of individual users. A user's access rights define the actions they are able to initiate within the system. For example, a student may have the ability to upload files to the front-end server. This upload may trigger the back-end server to update the analysis of the current assignment. If this were the case a student must have both the right to upload files, and the right to submit jobs to the back end. This function would determine if this was the case. Access rights may change dynamically during execution. This function should reflect any changes to access rights instantly.

4.28.2 Depends On

N/A.

4.28.3 Expected Inputs

The expected inputs of this function are a user ID and an action.

4.28.4 Expected Outputs

The expected output of this function is a token indicating whether or not the user may have performed the requested action.

4.28.5 Expected Side Effects

This function has no expected side effects.

Chapter 5

External Interface Requirements

5.1 User Interfaces

As of current, there is no built GUI standard. Following the creation of the front-end, there will be style guides and GUI standards which will be built and will mold the final product of the finished front-end portion. Tutorials for first time users accessing the user interface will be available (see 3.6 Web Tutorials). See the appendix for basic examples of how the the front-end will look for the user to interface with. More information and detail regarding these interfaces may be found in 3 System Features.

The user interface design will be made keeping in mind system status, user control and freedom, consistency, recognition, flexibility, efficiency, aesthetics, help and documentation. The system will be designed in order to keep errors to a minimum and be easily dealt with, and recovery from errors should be simplistic for the user.

The administrator would also have a Unix-like shell interface which they would interface with for applying permissions, etc.

5.2 Hardware Interfaces

The testing environment for this product will use Brock's Sandcastle server. The final product should be implementable, and access between software and hardware will be largely trivial. The only hardware used is the Sandcastle server itself, which allows for backend to be processed and only ever requests and returns data from and to the database. It also communicates with the front-end in order to update the status of the backend processes. Since the front-end will be web-based. there will be no communication with other hardware aside from personal computers accessing the website itself.

5.3 Software Interfaces

The web-based front-end will be using the standard HTTPS protocol and encryption. It communicates with the database through passing data to it for storage and for downloading data. This works through the presentation layer which uses Javascript to utilize the application/logic layer which will be using PHP in order to make contact with the data layer (the database).

As for front-end to backend communication, this will be done through sending requests and receiving responses to and from the Sandcastle server. This communication will be using TCP/IP and HTTP standard protocols. The backend API is the function whose purpose will be to submit jobs to the backend. This will be a batch process which effectively sends jobs.

The Sandcastle server uses PHP, and will be communicating with the database using requests. It will pull and push data to the database to save and load as necessary. The database and backend will be working together in order to fetch documents, update documents and retrieve analyses between one another.

5.4 Communications Interfaces

The communication interfaces which will be used or utilized in some way are web browsers and e-mail communication. The protocols which will be used are TLS 1.3 for cryptography, HTTP, TCP/IP, SSH for Sandcastle, and the built-in SQL server protocols.

Chapter 6

Other Nonfunctional Requirements

6.1 Performance Requirements

The system should be able to process 1000 documents in less than 8 hours, with the indexing per document not taking longer than 1 hour. Analyzed assignments should have a greater that 60% success rate on benchmark data. The system should be able to compare up to 10 years worth of assignments/documents.

6.2 Safety Requirements

In order to ensure student safety, the system must not provide a judgment or decision on the students' work. The system should provide similarity scores and the ability for the user to investigate cases of similarity between documents. All judgment and decisions should be left to the user investigating the similarities.

6.3 Security Requirements

All student information sent over a network should be encrypted (End to End Encryption). This ensures that in the case of interception, any intercepted information will be encrypted and will not be useful to the interceptor.
Data should only be decrypted when being processed by the backend, or when being stored into the database. This ensures that the only time the documents would not be encrypted would be when they are at their intended destination
The backend should not hold any data, therefore the backend will serve as a function. Since the backend will not store data, this ensures that in the case of a cyberattack, no student/school information can be intercepted. This statelessness on the backend ensures that cyberattacks will never result in privacy breach.
There should be a user lockout process, which locks the account if there are a number of failed logins. This lockout should be implemented in order to prevent a cyberattack through brute force.

6.4 Software Quality Attributes

The software should be easy for the intended user to operate, requiring little training to operate the basics functions of the front-end.
The system should have a failure rate no more than 0.023% ***DEFINE FAILURE***
The system should be easy to adapt to other universities, such that other universities can configure the system with their preferences and to comply with their policies and procedures.
The system should be well documented and easy to maintain for future developers.

The system should have easy-to-follow documentation for all intended users. Administrators should be able to easily understand and install the front-end portion of the software on their system. There should be simple yet comprehensive guides for the instructor/TA to navigate through all the features of the system.
The system should maintain a log of all interactions for moderation, performance monitoring, and maintenance purposes.
The front-end system should be useable in all browsers
The system should be able to handle large volumes of submissions without technical difficulties, and allow the user to know an estimated completion time.

Chapter 7

Implementation Specific Details

In an attempt to keep this document flexible, we had previously left out many details about the actual implementation of the system. This allowed for experimentation and flexible development. This chapter will specify the implementation details for the Water Fern system.

7.1 Front-end

With the goal of dynamically displaying student assignments and their relative similarity scores, the front-end of the system was built using standard HTML, CSS, and javascript. Additionally, the **jQuery** library was used to help with handling setting and readying cookies, which allowed for simpler development. Jquery also allowed shorthand access to plain javascript function, create objects in the page, make ajax calls.
Communication between front-end webpages is accomplished using cookies handled by Jquery. Any data stored in the cookies will be encrypted to avoid security concerns.

7.2 Back-end

Recall that the back-end server played the role of processing the documents submitted by the user. The back-end server used a number of different tools in order analyze and index the assignments for comparison.
A python library named **gensim** was used to automatically extract semantic meaning from data. This library includes a word2vec (word to vector) algorithm which assigns each keyword of a programming language a vector. This means that vectors used frequently in the text will be given a vector. Keywords that are often used within close proximity to each other will be assigned vectors which are closer to each other. These vectors can then be combined to represent longer strings of keywords, eventually allowing for comparison of blocks (such as methods, loops, classes, etc...) of code by vector comparison.
A parsing/lexing library called **ANTLR4** is used to analyze the assignment source files and generate parse trees. These parse trees represent the student source code at differing levels of granularity, with the root representing the entire code, and the leaves being the individual keywords. Using word2vec we are able to assign each leaf a vector, and combine the vectors to get the vector for the parent node. At the end of the process, each node in the parse tree is associated with a vector. Any two nodes that have vectors which are short distance away from each other can be said to be similar. In this way, we can compare all nodes of a parse tree to other nodes of other parse trees and determine which blocks of code are similar to each other.
In order to handle multiple requests in parallel, the python **Redis Queue** library is used to queue jobs for back-end processing. This allows the backend to process requests, the analysis of the documents, and IO bound processing, in parallel.

7.3 Database

All course/assignments information is stored on the front-end server in a database. The database implemented use **Postgres 10** with default configuration. To ensure secure communication between servers, Postgres was modifying to use ssl. The **psycopg2** library was also used for communication between the database and the servers.

7.4 Integration

In order to ensure that the 3 subsystems communicate properly, several tools were used to ensure the system's integrity. **Docker** is a tool for containerization, which allowed us to isolate the application from the operating system, and provide a prebuilt and custom environment for the application. These features will be integral to the eventual scaling of the software, as it will allow for simple deployment at scale. At time of development Docker is used to help ease the process of distributing our system, but will lend nicely to the eventual need to scale the product.
The **Flask** python library was used to help communication between the front-end and the database. Flask is always listening at specific parts and when requests are sent, flask can dynamically generate content based on the request. This allowed the front-end to make requests for information to Flask endpoints, which then returns the information to the front-end. Different flask endpoints can generate different data based on the requests, allowing for flexible content generation.
Nginx was used as the web-server for Flask. Nginx handles the outside requests, passes them along to Flask for content generation, and then passes the content back to the user. Nging was configured to use HTTPS in order to make all communication between the servers secure, minimizing the risk of data interception during server communication. In addition, **UWSGI** was used as middleware to facilitate communication between Nginx and Flask.

Chapter 8

Glossary

- **Similarity** - A similar feature or aspect; Used to describe reports that may potentially be plagiarized
- **Plagiarism** - The practice of taking someone else's work or ideas and passing it off as your own; It holds a zero-tolerance acceptance in University
- **Myriad** - A plethora
- **MOSS** - Measure of Software Similarity, a software system created by Stanford which is currently the widespread software being used for plagiarism and similarity detection
- **Linux** - An open-sourced operating system modelled on UNIX
- **FIPPA** - The Freedom of Information and Protection of Privacy Act, which is Ontario's law. It essentially states that all personal information handled online is properly encrypted and handled, so as to not have any breaches in personal information
- **AODA** - The Accessibility for Ontarians with Disabilities Act, which denotes that proper channels are given for disabled people which allow them ease of access to the product at hand
- **W3C** - The World Wide Web Consortium, which is an association that defines basic minimum requirements or standards for online products
- **Front-end** - Anything denoting the portion of the computer system to which the user interacts with
- **Back-end** - The portion of the computer system to which the user does not interact with; Storing and manipulation of data gets handled within the back-end
- **System Administrator** - The user responsible for building and maintaining the Water Fern system
- **CSV** - Comma Separated Values, a delimited text file type
- **Java** - A general-purpose programming language designed to be able to produce programs meant for any system
- **AES** - Advanced Encryption Standard, a symmetric block cypher which is considered the standard for back-end file encryption
- **GUI** - Graphical User Interface, the portion of the system the user physically can see and interact with on screen
- **HTTP/S** - Hyper Text Transfer Protocol Secure, the underlying protocol used by the World Wide Web which defines how messages are formatted and submitted (Secure meaning it is a secure version of HTTP)
- **TCP/IP** - Transmission Control Protocol/Internet Protocol, which is used for governing the connection of computer systems to the internet

- **PHP** - Hypertext Preprocessor, a general-purpose programming language designed for web development
- **TLS** - Transport Layer Security, cryptographic protocols designed to provide communications security over a secure network
- **SQL** - Standardized Query Language, a language intended to be used for database communications
- **End to End Encryption** - When both ends of a connection are encrypted such that neither system at the end of the connection is subject to an invasive attack
- **CSS** - Cascading Style Sheet, a style sheet language used for describing the presentation of a document in a markup language (such as HTML)
- **HTML** - Hyper Text Markup Language, a standardized system for tagging text files to achieve font, color, graphic, and hyperlink effects on World Wide Web pages
- **Nginx** - A free, open-source, high-performance HTTP server and reverse proxy
- **Flask** - A micro web framework written in Python

Appendices

Appendix A

Analysis Models

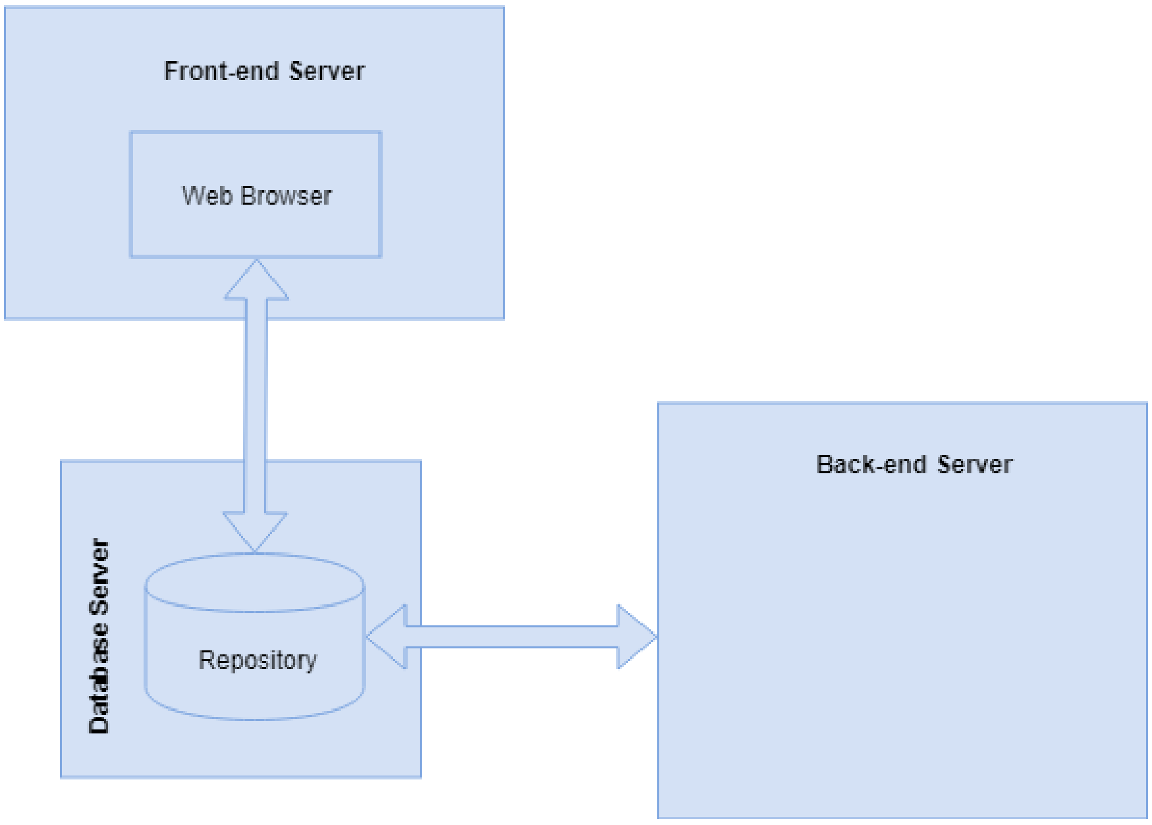


Figure A.1: System Diagram

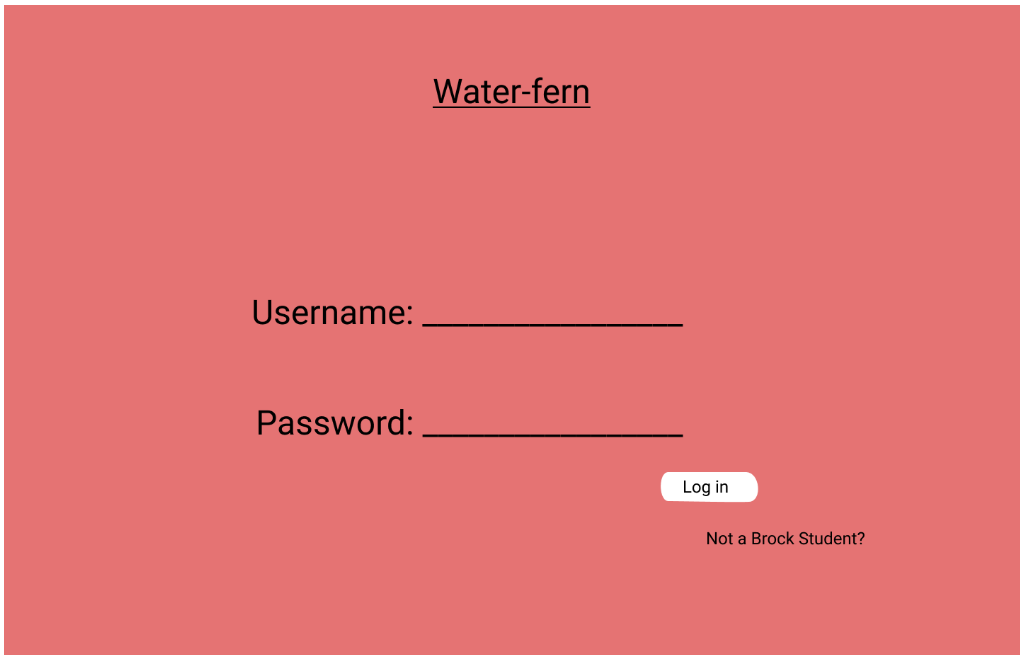


Figure A.2: General Login for all users

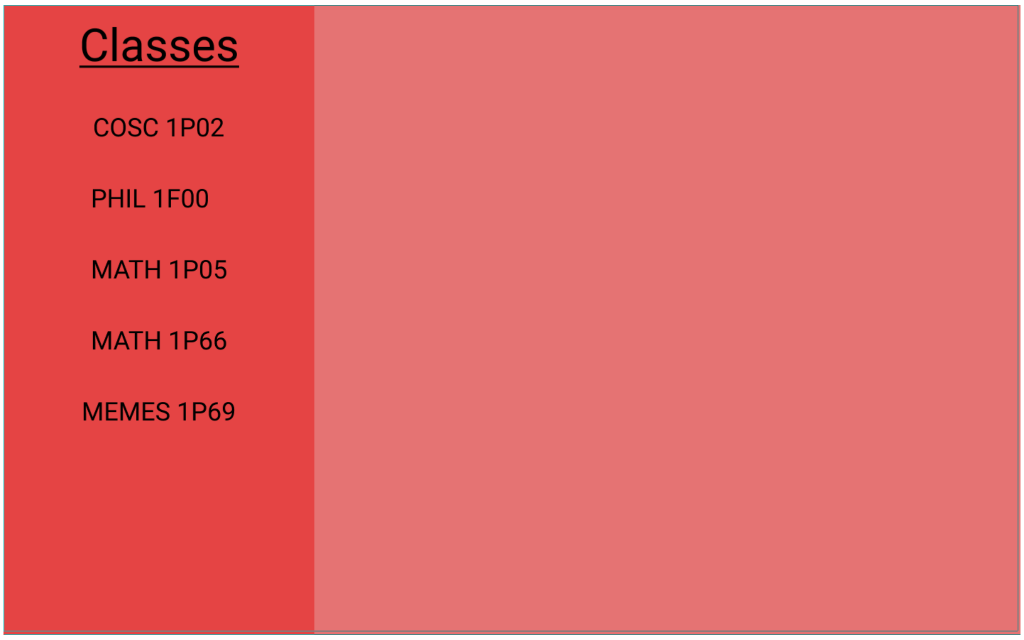


Figure A.3: Student Class Selection

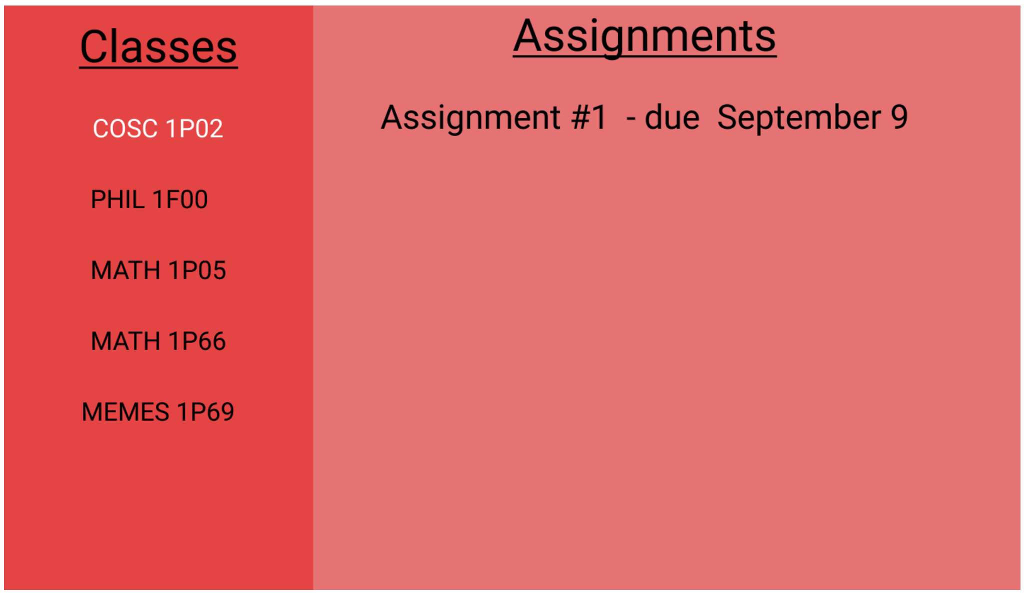


Figure A.4: Student Assignment Selection

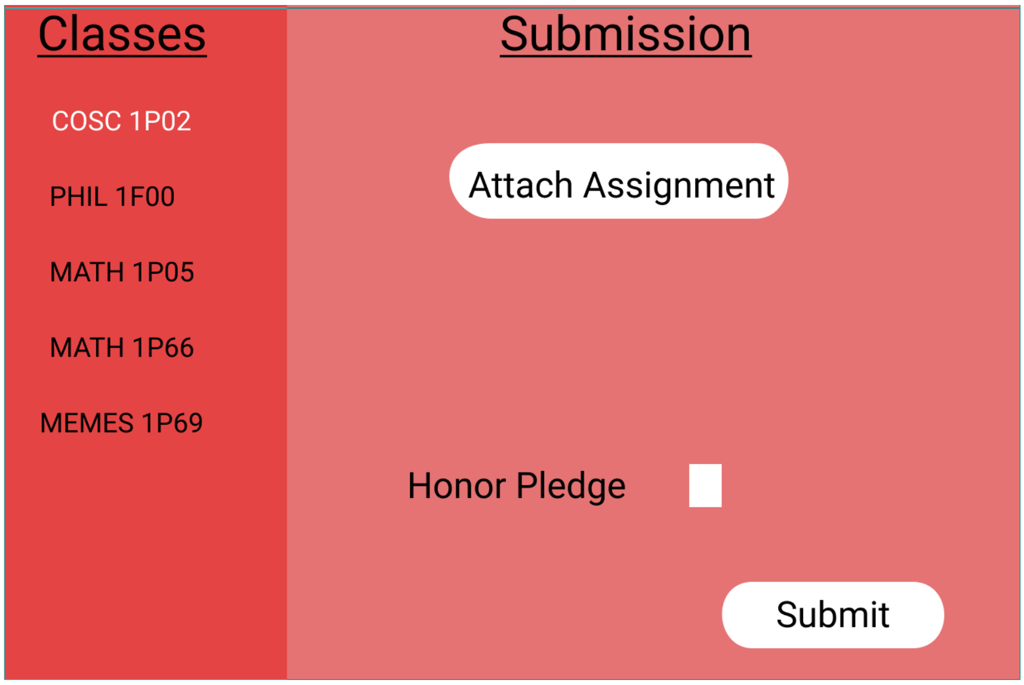


Figure A.5: Student Submission

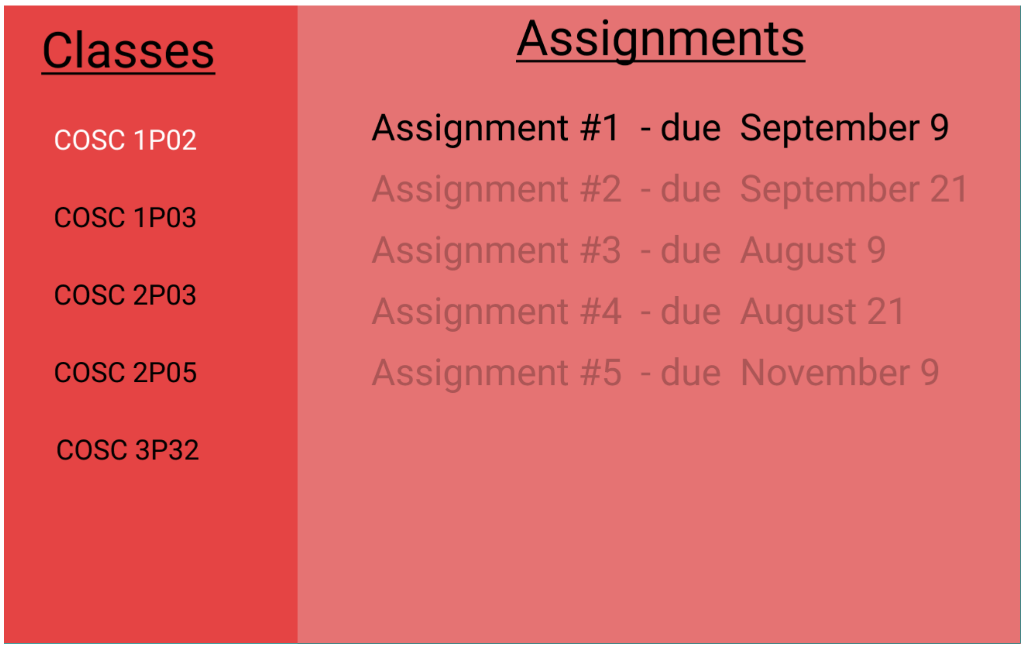

Classes
COSC 1P02
COSC 1P03
COSC 2P03
COSC 2P05
COSC 3P32
Assignments
Assignment #1 - due September 9
Assignment #2 - due September 21
Assignment #3 - due August 9
Assignment #4 - due August 21
Assignment #5 - due November 9

Figure A.6: Instructor Assignment Selection with inactive assignments

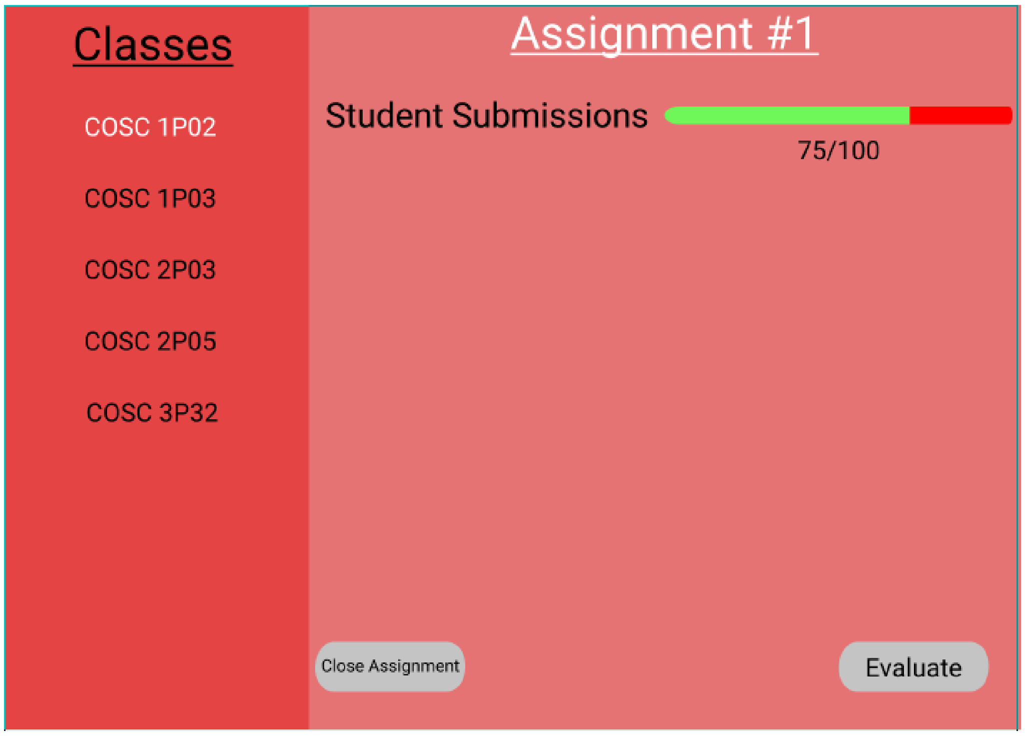
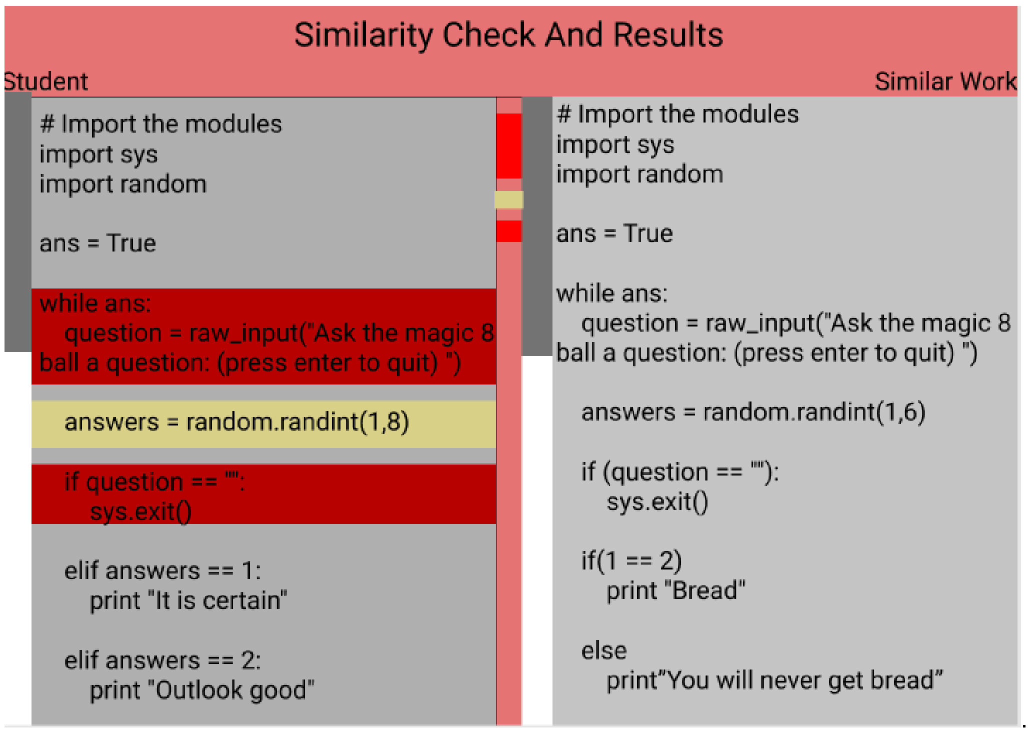


Figure A.7: Instructor Assignment Progress



```
# Import the modules
import sys
import random

ans = True

while ans:
    question = raw_input("Ask the magic 8
ball a question: (press enter to quit) ")

    answers = random.randint(1,8)

    if question == "":
        sys.exit()

    elif answers == 1:
        print "It is certain"

    elif answers == 2:
        print "Outlook good"
```

```
# Import the modules
import sys
import random

ans = True

while ans:
    question = raw_input("Ask the magic 8
ball a question: (press enter to quit) ")

    answers = random.randint(1,6)

    if (question == ""):
        sys.exit()

    if(1 == 2)
        print "Bread"

    else
        print”You will never get bread”
```

Figure A.8: Similarity Checking

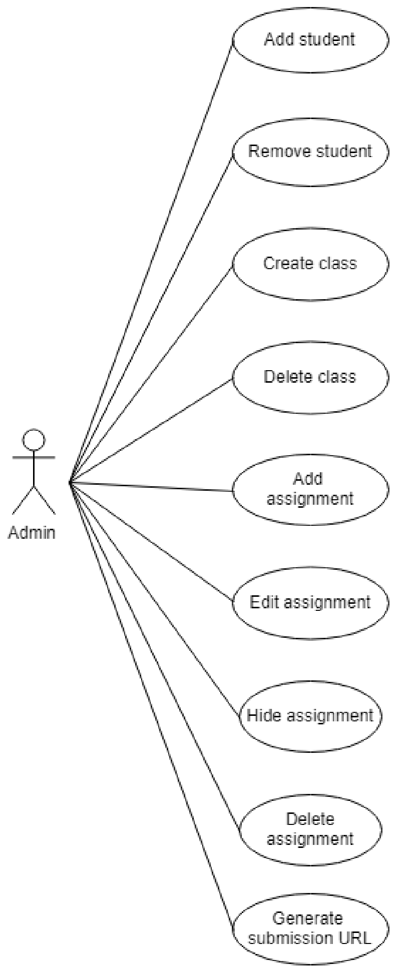


Figure A.9: Administrator Use Case

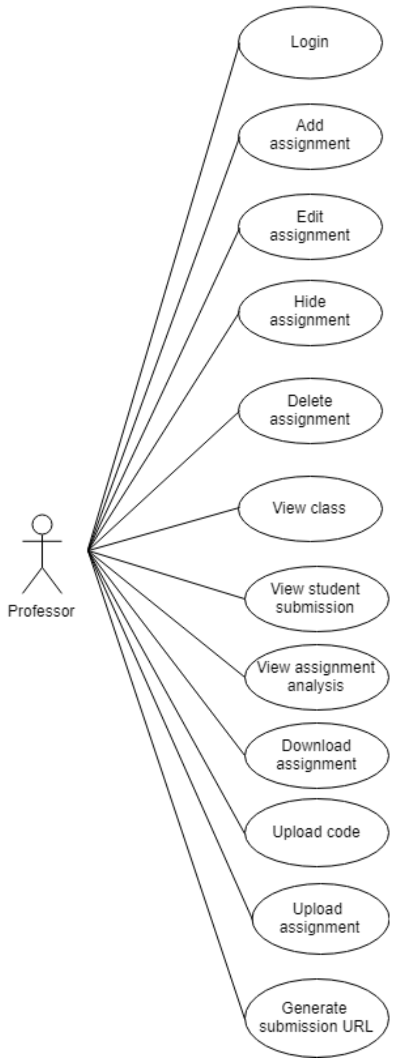


Figure A.10: Professor Use Case

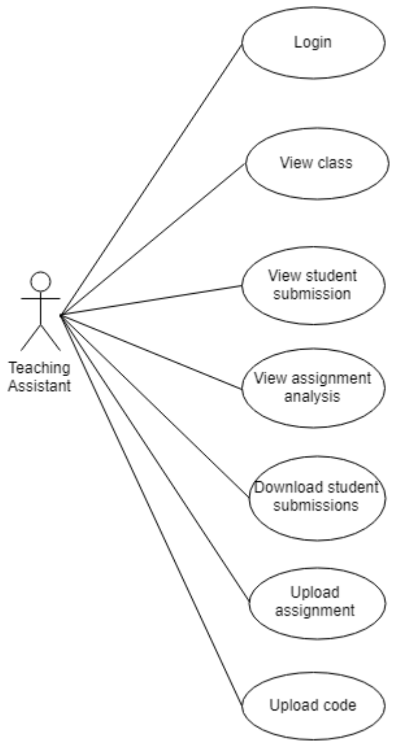


Figure A.11: TA Use Case

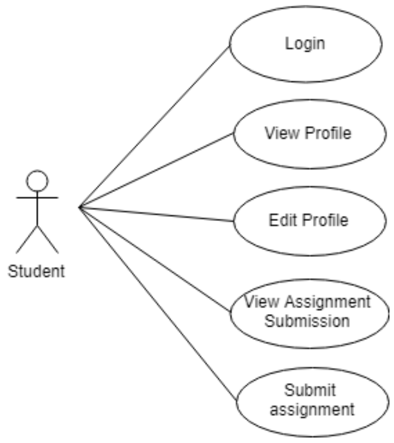


Figure A.12: Student Use Case

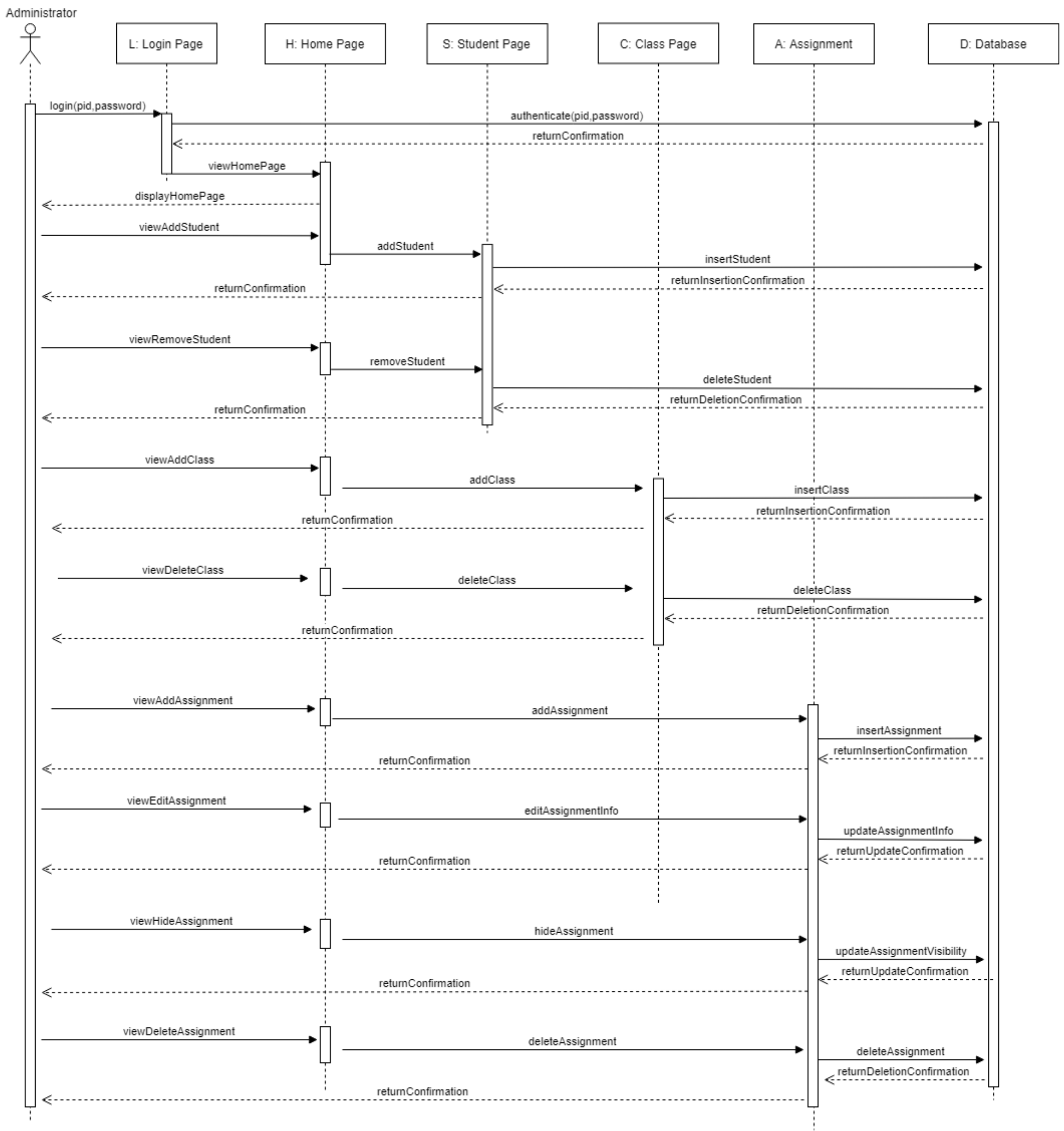


Figure A.13: Administrator Sequence Diagram

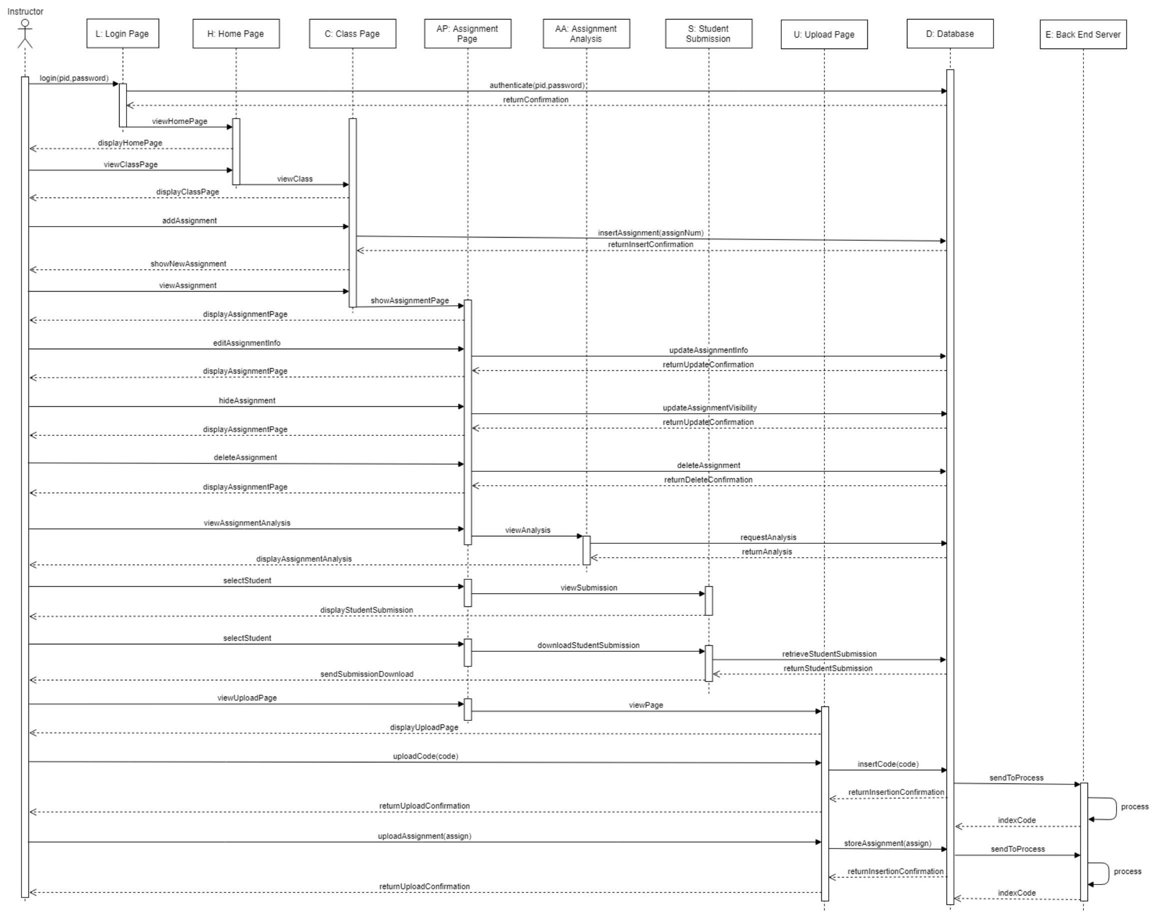


Figure A.14: Professor Sequence Diagram

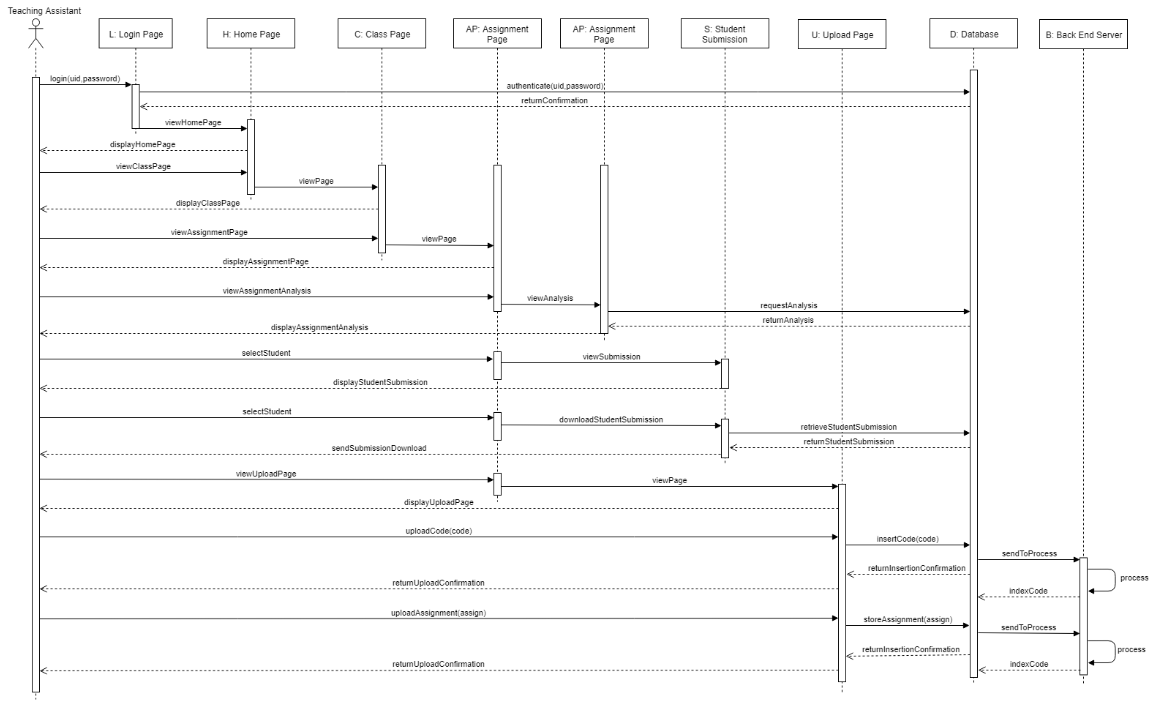


Figure A.15: Instructor Sequence Diagram

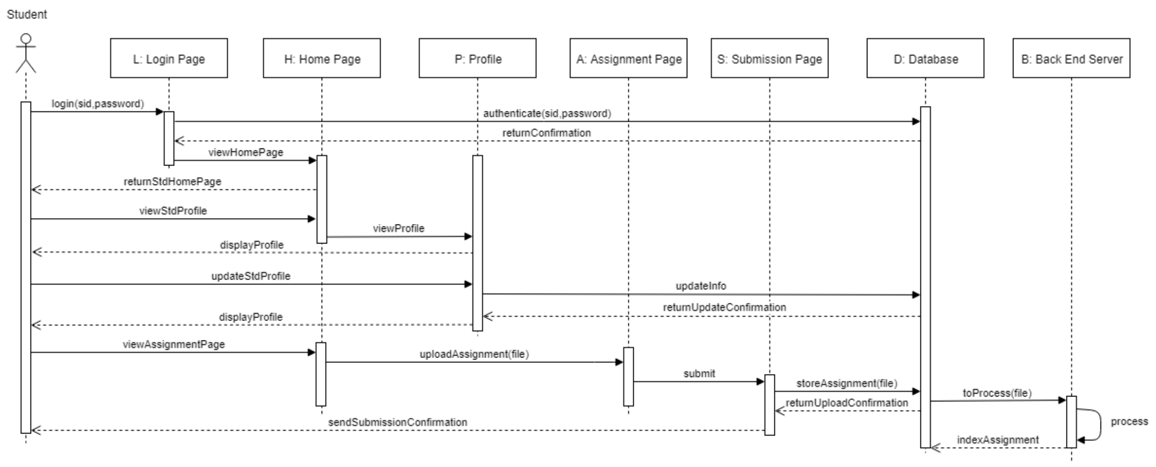


Figure A.16: Student Sequence Diagram

Appendix B

Test Results

Below are the results from the test suite developed for the system.
Note: The two FAILS are associated with features which have not yet been fully implemented.

```
 ***
Testing to ensure that found associations can be retrieved. ... ok
Testing that associations between the submit files from the flat zip were stored in the DB ... ok
Testing to ensure that the endpoint for retrieving associations rejects malformed requests. ... ok
Testing for malformed input to getSubmission after submit from the flat zip ... ok
Geting submission with id 1 from the flat zip ... ok
Testing that indexes were generated on each file from the flat zip ... ok
Testing that the submit files from the flat zip were stored in the DB ... ok
Testing to ensure that found associations can be retrieved. After submission from the non-flat zip ... ok
Testing that associations between the submit files were stored in the DB ... ok
Testing to ensure that the endpoint for retrieving associations rejects malformed requests. ... ok
Testing for malformed input to the getSubmission page ... ok
Geting submission with id 1 from the non-flat zip ... ok
Testing that indexes were generated on each file from the non-flat zip ... ok
Testing that the submit files from the non-flat zip were stored in the DB ... FAIL
test_invalid_source_code_java (Tests.TestSubmissionOfBrokenCode) ... FAIL
test_missing_assignment_ID_field (Tests.TestSubmissionOfBrokenCode) ... ok
test_missing_user_ID_field (Tests.TestSubmissionOfBrokenCode) ... ok
test_get_assignment_list (Tests.TestVisitingFrontEnd) ... ok
test_get_assignment_list_malformed_input (Tests.TestVisitingFrontEnd) ... ok
test_get_list_of_classes (Tests.TestVisitingFrontEnd) ... ok
test_get_offerings (Tests.TestVisitingFrontEnd) ... ok
test_https_redirect (Tests.TestVisitingFrontEnd) ... ok

======================================================================
FAIL: Testing that the submit files from the non-flat zip were stored in the DB
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/cody/Documents/school/cosc/4f00/github/Water_Fern/VersionAlpha0.0.1/Tests.py", line 173, in tes
    self.assertTrue(totalSubmissions == 2, "There should be 2 submissions in the DB not " + str(totalSubmission
AssertionError: False is not true : There should be 2 submissions in the DB not 4

======================================================================
FAIL: test_invalid_source_code_java (Tests.TestSubmissionOfBrokenCode)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/cody/Documents/school/cosc/4f00/github/Water_Fern/VersionAlpha0.0.1/Tests.py", line 254, in tes
    cls.assertTrue(str(r.content).find("Invalid Code") != -1, "Invalid code should have been detected")
AssertionError: False is not true : Invalid code should have been detected
-------------------- >> begin captured logging << --------------------
requests.packages.urllib3.connectionpool: DEBUG: Starting new HTTPS connection (1): 0.0.0.0
requests.packages.urllib3.connectionpool: DEBUG: https://0.0.0.0:8001 "POST /api/v1/uploadsubmission HTTP/1.1
--------------------- >> end captured logging << ---------------------

----------------------------------------------------------------------
Ran 22 tests in 60.593s

FAILED (failures=2)
```

Appendix C

Change Requests

References

[1] Alex Aiken. A system for detecting software similarity, December 2018.

[2] Tricia Bertram Gallant, Nancy Binkin, and Michael Donohue. Students at risk for being reported for cheating. *Journal of Academic Ethics*, 13(3):217–228, 2015.

[3] Aaron Hutchins. Universities new rules for cheating, March 2017.

[4] Hermann A Maurer, Frank Kappe, and Bilal Zaka. Plagiarism-a survey. *J. UCS*, 12(8):1050–1084, 2006.