Requirements Specification

For

Measurement Of Code Duplication - MOCD

Version 1.0

Prepared by Jenson Hung, Liam Marcassa, Tina Pazaj, Jesse Treleaven, Juan Manuel Vallecilla, Ji Wang

4F00 Dr Java Analysis & Design

February 4, 2019

TABLE OF CONTENTS

Introduction

Purpose

This requirements specifications document outlines the requirements for the MOCD (Measurement Of Code Duplication) plagiarism detector. This new detector will be an upgrade to the current plagiarism detector being used for Brock CS students also known as MOSS (Measurement Of Software Similarity). MOSS is a plagiarism detector for Java, C, and C++ code. This requirements specifications document specifies requirements already implemented as well as requirements for future releases.

Intended Audience and Reading Suggestions

The intended audience for this software is for school faculty--primarily for teachers and students. The secondary audience is intended for developers, testers, and writers. The reader is assumed to have basic knowledge of common technical terms such as server, client, GUI, web, etc. The purpose of this requirements specification document is to define the project scope and all requirements needed for the audience to understand and manage the software.

Project Scope

The scope of this project is to be an improvement to the MOSS plagiarism detector. The software will efficiently and quickly detect plagiarism and the goal is to minimize the amount of plagiarism done by students. This project is meant to be free to use, however it could be monetized in the future. The system and software outlined in this document is not intended for ongoing development, in fact it is an iteration / evolution of an existing system. Even so, best practices are being applied should future development become a possibility

References

1. I. Sommerville, *Software Engineering*. Boston: Pearson, 2016.

2. S. Schleimer, D. S. Wilkerson, and A. Aiken, “Winnowing,” *Proceedings of the 2003 ACM SIGMOD international conference on the Management of data - SIGMOD 03*, 2003.

Overall Description

Product Perspective

This product is an improvement of the current the MOSS plagiarism detector. The goal of this project is to increase the ease of Stanford’s original system. It will not rely on any of the original components from Stanford, and should allow a much broader user base with less technical expertise. This improved product consists of a central server on which the plagiarism detection is being performed and a local repository of files. It also consists of two related web interfaces; one for students, one for teachers.

Product Features

This software contains several significant functions and user-friendly features. The main functions consist of students submitting files to a course, teachers assigning students to a course, creating courses, and managing/adding/deleting course info. Teachers will also be able to easily send and review all submitted assignments for plagiarism and be able to supply material which should be excluded from comparison, and further, be able to tune the detection granularity. On plagiarism being detected, the system will provide results in a convenient interface for analysis. Data sent off-site is completely anonymized to preserve student privacy and it is not stored for any great length of time to ensure student copyright is protected. The software will make maximum availability and transparency a priority, and most importantly ease of use.

User Classes and Characteristics

Students:

This user class is comprised of any person attending a course that involves code submission. This user is assumed to have basic knowledge of common technical terms as well as knowledge of course guidelines and school policy as specified by the teacher at the beginning of the course. The privilege level of this user will be low but the data they submit will be highly secure. This user should only be able to submit assignment files as required and allowed by the teacher., This software is intended to be highly user-friendly so the user is not required to have training prior to using the system.

Teachers:

This user class is comprised of any person teaching a course at an academy and will require students to submit code assignments in Java, C, and/or C++. It is assumed that this user has a moderate level of knowledge in common technical terms as well as knowledge of course guidelines and school policy. This user has both high security and high privilege levels on the software. They will have access to several features and the ability to remove/add course and students. All information submitted by this user will be secure and confidential unless they specify otherwise. This user should have some training prior to using the system. Tutorials and FAQs provided on the site should be sufficient.

Admins:

This user class is comprised of any person installing, upgrading, or managing the accounts of other users of MOCD. People who manage the detection server are not included in this group; only people who work with school servers. These users are assumed to have a high degree of technical knowledge. They have the highest level of privilege and take on the greatest risk.

Developers:

This user class is comprised of any person who is writing software for MOCD or managing the remote (detection) server. They are expected to be familiar with coding practices. They take on a great deal of responsibility and are tasked with ensuring other roles can perform their job well.

Operating Environment

From the teacher and student perspective, MOCD is accessible through all major browsers. At minimum, it should provide a consistent experience across: Chrome, Safari, Firefox, and Edge, with even more browser support preferred. This list defines a sufficient test environment for the teacher and student roles as it covers the vast majority of the user experience. Portals should be navigable on mobile phones, however this is not high priority.

Developers and Admins will not be interacting with MOCD through a web interface, but instead through scripts or raw code. As the system will be programmed primarily in Java and React, choice of OS is largely left to the individual (as both technologies are cross-OS). The remote detection server will be hosted on a Unix-like environment, and the school server may be required to be hosted on the same, but that is to be decided.

Design and Implementation Constraints

Care should be taken to ensure the results of plagiarism analysis are correct, in order to prevent both false positives and negatives, but teachers are to be reminded that original sources are the only truth. MOCD will not provide any methods for grading or mark assignment. Even though security and anonymity is a high priority for MOCD, student grades are considered to be too confidential. The system will be licensed so developers do not own any data which is submitted by teachers to the remote detection server. Student data must be anonymized.

Developers will be required to follow the guidelines and plan set forth in this document, however the requirements may change as limitations of implementation are discovered. Any deviations from or amendments to the requirements made during development must themselves be documented. The developers may use any software development methodology they see fit, at the discretion of project management.

Students will not be able to create assignments, submit files for duplication detection, or view the plagiarism results. These limitations stem again from the a risk benefit analysis of confidentiality.

User Documentation

User documentation will cover installation of the system, features, tutorials, help, and release notes, all provided online. The completeness and coverage of these documents are to be determined, as in this particular project the working code takes slightly higher priority. In the absence of any further documentation, this paper may be distributed. In addition, it is intended that the resulting system supports self-documentation: the controls and operative sequences can be used and understood when teachers and students first interact with the system.

Assumptions and Dependencies

This document assumes that all users are familiar with using a computer, including: internet browsers, keyboards, and mice. The software constraints are that the system is only able to compare Java, C, and C++ code, further languages may be added given further development time. Dependency is FF(Finish to Finish). All code submitted must be anonymized before being compared. While in development, the system will be tested using data generated by the developers. It assumed that code being compared can be compartmentalized by a teacher into assignments. This is not to say that a teacher could parcel any arbitrary group of files into an assignment, but it is the use case which is being designed for.

System Features

Installation and Role Creation

This is a high priority feature. A system administrator should be able to install and deploy a school server which can connect to the plagiarism detection server, remain secure, and facilitate multiple users simultaneously.

A system administrator downloads an executable installer from online, and reads the accompanying documentation. The exact detailer of such an installer are to be decided, but may consist of a pre-packaged binary or series of scripts with accompanying instructions. When connecting to the detection server for the first time, credentials must be provided. The administrator is able to create accounts for two types of role: teachers and students. The credentials for these roles are emailed to some address, and must be changed after first login. Credentials may be revoked by the administrator at any time. Student accounts may be created via a csv file with a list of names, SIDs, and emails.

A larger number of roles with corresponding access patterns would enable more people to use the system safely, however the teacher and student roles are sufficient. The system must be able to match credentials to access levels. Credentials must be malleable and secure. The steps for setting up the system must be tested to be working.

Course Creation and Deletion

This is a high priority feature. Teachers must be able to create and delete courses, with full ownership over the pertinent files.

A teacher is able to create a course, and must provide a unique combination of: course code, name, year, and semester. Further, a course duration is encouraged, but not required. The system will confirm or deny the request. If successful, the teacher may then select the course from a list of courses they have created. Upon selecting a course, the teacher may choose to delete it. After confirming deletion, students are automatically unenrolled and submitted assignments relating to that course are marked for deletion.

The teacher must have login credentials and authorization to access course creation and student enrollment services. The system must keep track of courses already created and be able to compare course information. Teachers are able to view only courses which they have created and no courses created by other teachers. This may be a nuisance in practice, and some sharing feature may be added later. Any and all documents submitted to a course must be known and stored in an organized directory.

Enrolling Students

This is a high priority feature. Teachers must be able to enroll students in any course they are teaching via the GUI or a csv file, and able similarly to remove them.

After selecting a course, a teacher may choose to enroll students. They may use a search bar to find students by name or SID. In this way, individual students may be enrolled in a specific course. Alternatively, a teacher may choose to upload a csv file with student names and SIDs (exact format to be specified, but must include at minimum SID). Upon successfully parsing the file, the system will display all students to be enrolled in the course and warn of any unmatched credentials. Students are able to view all courses which they are currently registered in through their own web portal. Instead of enrolling students, a teacher can remove (unenroll) a student via the GUI, but not by uploading a csv. In addition, there is an option to remove all students from the course. At every stage in this process, there is an option to cancel, in which case no further actions will be taken and the web interface will return to an appropriate page.

The system must be able to handle file uploads from a remote disk, and be able to parse csv files. Students must be known to the system by both name and sid. Duplicates and missing students should be detected. Unenrolling students does not delete their submitted assignments.

Assignments

This is a high priority feature. A teacher can create an assignment belonging to a course, which students can then submit files to.

A teacher may select one of the courses they are teaching. They are subsequently provided the opportunity to view current and past assignments or create new assignments (among other options). On viewing past assignments, a similarity report may be available, discussed later in this section. On viewing current assignments, a list of students and their corresponding files are present. The teacher may decide to submit this list for duplication detection (along with some other files, similarly discussed later in this section). Finally, the teacher may choose to create a new assignment. A unique assignment name (within that course) must be provided on the subsequent screen, along with options for opening and closing dates, resubmission limits, file type constraints, and file size limits. Once created, the teacher is able to select the assignment from a list of assignments (per course) and view all student submissions (their name, sid, and a list of files). After creation, the teacher is able to modify assignment fields at any time.

After a student has submitted files for an assignment, they are able to see when the submission took place and file names included in the submission but not the original files. This encourages students to keep their own copies and prevents them from downloading past assignments. If a student wishes to re-submit to an assignment (for example, if they found a

mistake in their code before the assignment's closing deadline), their old submission will be overwritten (deleted).

The name of all assignments within a course must be known. The teacher may close the assignment at any time, in which case no students are allowed to submit any more files. Students are able to submit files with similar names without the software mixing them up. Teachers are able to delete an assignment at any time, in which case all associated files will also be deleted. Student's are able to re-submit up to resubmission limit times, with new files completely overwriting old files: old files are marked for deletion. Students are able to see the names of their files most recently submitted, and timestamp of when that occurred, but not the contents of the files.

Detection Process

A teacher can submit a all files relating to an assignment to the remote server for plagiarism detection.

When an assignment has been selected, the teacher may choose to submit all associated files to the remote detection server. They are able to include one or many files which are to be excluded from plagiarism detection (common, boilerplate, or provided code). In addition, they are able to submit files which themselves should not be rated for plagiarism but which should be matched against other files; solutions posted online could be included in this way. They may also choose how particular the detection is: whether three consecutive words must match or three lines. The details of this granularity are to be decided but some granularity scale should be available. A copy of all files are made which are then stripped of student names and SIDs before being compressed and sent over the network to the remote server. The remote server un-compresses the bundle of files, runs a proprietary algorithm to determine the presence and level of plagiarism between files submitted. A report is returned to the school server detailing if and to what extent plagiarism was detected.

If multiple assignments could be submitted at the same time, plagiarism from year to year or between assignments could be detected. This is a lower priority feature that would enhance the usefulness of the system.

This feature requires that ownership of files is maintained in a repository on the school server. A suitable compression and plagiarism detection algorithm must be used (see the "Winnowing" paper in the references section for an idea).

Plagiarism Review

The extent of detected plagiarism should be available to a teacher on a per-student basis. This is a high priority feature. A high-precision report would allow decisions to be made

more quickly, however extra detail is of lower priority as this feature would require significantly more work. A single “similarity estimate” on a per-student basis is the only absolute requirement.

After a teacher has submitted one (or possibly several) assignments to the remote server, they will be able to view a plagiarism report. The exact details of this report are to be decided, but at minimum they must be available through the web interface and indicate what percentage of each student’s submission was found to a copy of another submitted file. A better interface would provide a per-student percentage, their historical percentages, and an option to view percentages on a per-file basis. When viewing a single file, individual passages which were found to be plagiarized would be highlighted, and link to a list of files where similar passages were found.

The report should be returned within a decent time from the request, maybe within 24 hours. Teachers should be able to inspect the original files to corroborate the system’s results.

Web Scraping / External Verification

This is a low priority feature. In order to increase the quality of detection, the remote server could scrape the web for common snippets of code and match them against any submitted files.

Before submitting to the remote detection server, teachers could check off whether they want MOCD to match against scraped files or not. This would increase the server load and the search space may prove to be too vast for returning reports in a timely manner; testing would be required. The returned plagiarism report would include links to similar code found online.

This would require the remote server to handle a much larger size of data, and long-term storage of that data. Care must be taken to construct a spider which does not overburden websites and scrape redundant or irrelevant passages.

External Interface Requirements TODO

User Interfaces

Hardware Interfaces

The MOCD system relies on two servers communicating over the internet, and several users connecting to the school server, again over the internet. The details of networking are out of scope of this document. The two servers may be whatever type, of sufficient power and capacity to handle the traffic.

Software Interfaces

Three existing pieces of software are crucial to the success of this system: Java, the React framework, and the MySQL database management system. The details of each interface and their importance is outlined below.

Java will underpin much of the algorithmic workload on the remote server. It is a general purpose language and runs quickly, facilitating interaction with other interfaces and ensuring users will not be frustrated by slow results. The developers familiarity with Java should decrease time to release.

The React web framework, or one similar, is needed to handle the bulk of user interaction. It must understand user roles and be capable of responding to requests dynamically. There must be a mechanism for translating the plagiarism report from the remote server into one or more web pages.

The final interface is a DBMS, or database management system. This will organize data stored on both servers, temporarily in the case of the remote server and permanently on the school side. MySQL is a common choice and has been around for some time.

Communications Interfaces

Two groups of interfaces are important, depending on the user role; teachers and students will access the system from a web interface, while the administrator will primarily use a command line.

For teachers and students, a web portal must be provided. This promotes ease of access and it is assumed to be a familiar environment for all users. The web portal should be tested for continuity and acceptance across common browsers. At the time of writing, browsers under test should include, at minimum, Edge, Chrome, Safari, and Firefox, preferably testing more. The experience should be similar across platforms, applications, environments. It is up to the discretion of the developers to ensure this is the case; they must have a test plan in place. With the increase in mobile devices, ensuring that the web portal may be navigated efficiently on smaller touch screens is of medium priority. There is a high development cost with limited benefit, as all users are assumed to have reasonably frequent access to a computer with a large screen. As this interface is hosted on the school's server, access is at their discretion.

Administrators need to be able to install the system and manage user accounts. A graphical, web-based interface has already been explored above. This interface may not be suitable for administrative purposes, as permissions are not always easy to grant through a network. As such, the minimum requirement is a command-line interface, with a series of scripts used to accomplish tasks.

Other Nonfunctional Requirements TODO

Performance Requirements

Firstly, availability of the school server is entirely up to the school. The remote server needs downtime for maintenance and upgrades which will be scheduled and made public. The software on the school server should be reasonable responsive for given amounts of traffic and hardware. The plagiarism detection algorithm on the remote server should run in reasonable time, and the integrity of results should not compromised.

Safety Requirements

Role separation is important for safe data handling: students should not have the same level of access as teachers, delimited by their username and password. This system provides no mechanism for assigning grades to students or their assignments and takes no responsibility of the consequences should the system be modified to do so. Irreversible actions should require a two-step confirmation process.

Security Requirements

Software running on the school server makes every effort to prevent compromising the identity of those using it, and maintaining the secrecy of submitted files from those who should not have access. When first setting up a school server, the administrator is provided with credentials for connecting to the remote server. These credentials are created on a per-school basis, so actions may be tracked back to the school on the remote server side. In addition, this prevents unauthorized access to the duplication detection process.

Software Quality Attributes

To ensure adherence to the above policies and of overall code quality, unit tests should be written and run at regular intervals. This will document the effort which has been put into security and provide an additional layer of oversight; flaws or bugs in the code may be detected earlier. In addition, all functions in the code need to be commented with an explanation of their parameters, computations, and outputs.

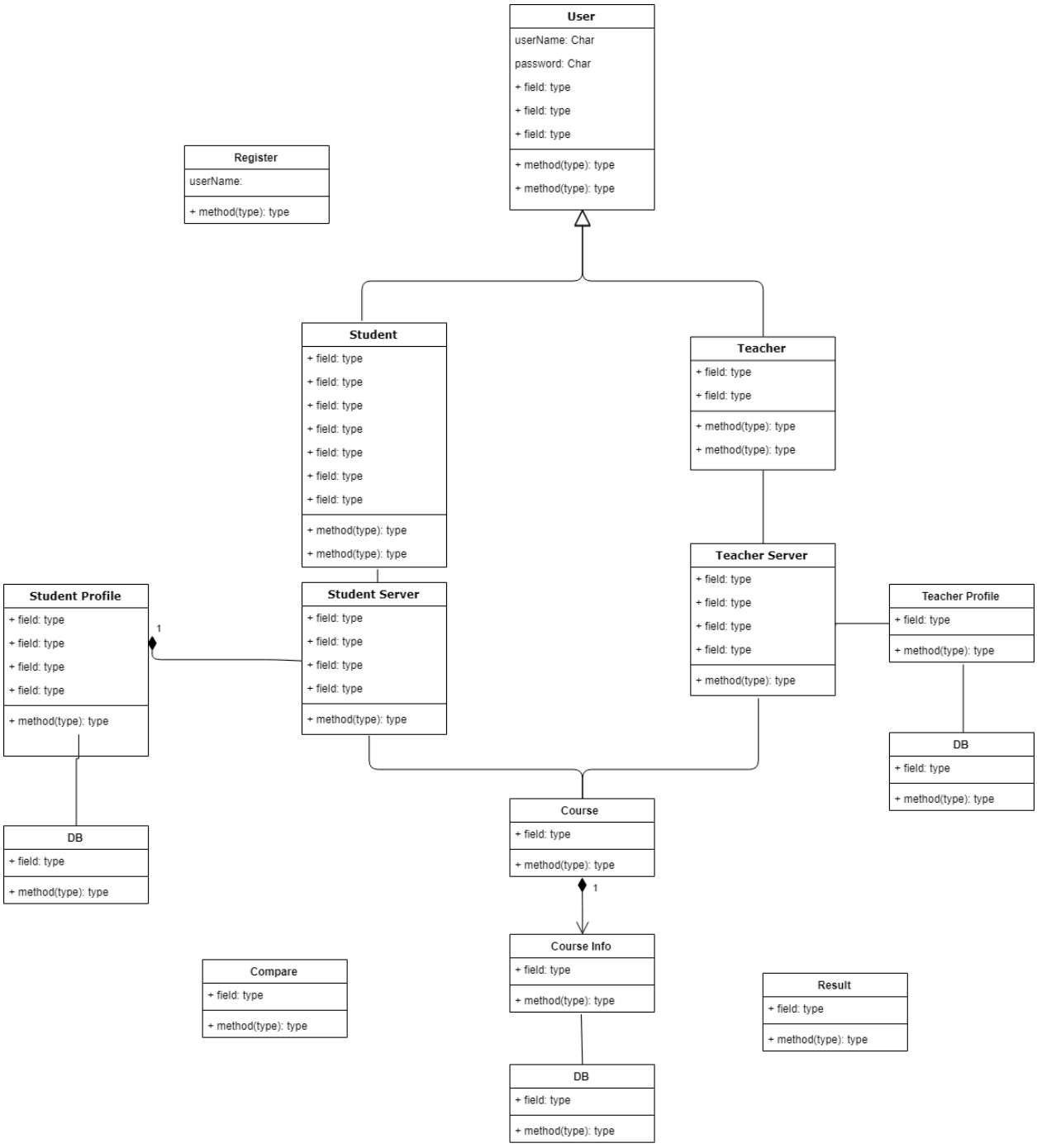
Other Requirements

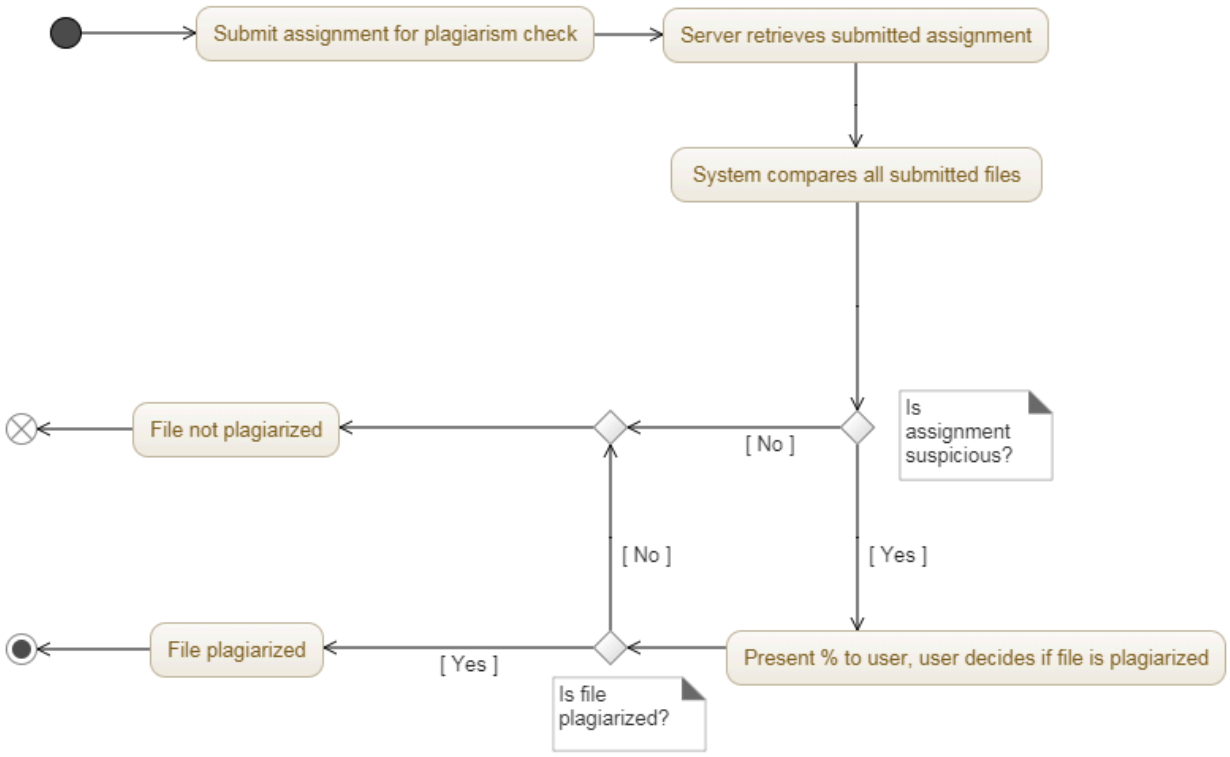
Appendix A: Glossary TODO

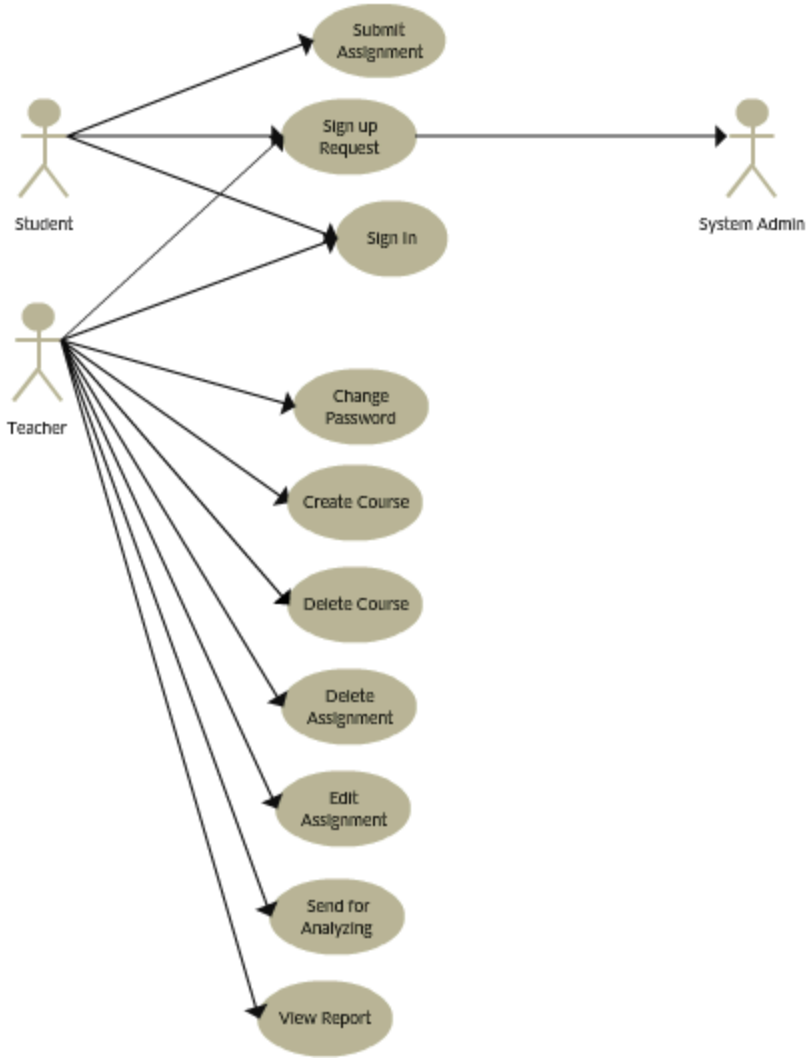
MOSS
MOCD
GUI
Server, school/database, remote/detection
Java
React
SID

Appendix B: Analysis Models ??? Diagrams? TODO

Some description of each of the diagrams. Maybe a title for each of them.


User
userName: Char
password: Char
+ field: type
+ field: type
+ field: type
+ method(type): type
+ method(type): type
Register
userName:
+ method(type): type
Student
+ field: type
+ field: type
+ field: type
+ field: type
+ field: type
+ field: type
+ field: type
+ method(type): type
+ method(type): type
Teacher
+ field: type
+ field: type
+ method(type): type
+ method(type): type
Teacher Server
+ field: type
+ field: type
+ field: type
+ field: type
+ method(type): type
Student Profile
+ field: type
+ field: type
+ field: type
+ field: type
+ method(type): type
1
Student Server
+ field: type
+ field: type
+ field: type
+ field: type
+ method(type): type
Teacher Profile
+ field: type
+ method(type): type
DB
+ field: type
+ method(type): type
DB
+ field: type
+ method(type): type
Course
+ field: type
+ method(type): type
1
Course Info
+ field: type
+ method(type): type
Compare
+ field: type
+ method(type): type
Result
+ field: type
+ method(type): type
DB
+ field: type
+ method(type): type


Submit assignment for plagiarism check
Server retrieves submitted assignment
System compares all submitted files
Is assignment suspicious?
[No]
File not plagiarized
[Yes]
[No]
Present % to user, user decides if file is plagiarized
Is file plagiarized?
[Yes]
File plagiarized



Appendix C: Issues List - TODO yes