ASHESI

ASHESI UNIVERSITY

AUTOGRADING AND DETECTING PLAGIARISM IN STUDENT PROGRAMMING ASSIGNMENTS

APPLIED PROJECT

B.Sc. Computer Science

Alex Waweru

2019

ASHESI UNIVERSITY

Autograding and detecting plagiarism in student programming assignments

APPLIE PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Alex Waweru

April 2019

DECLARATION

I hereby declare that this [capstone type] is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate’s Signature:

…………………………………………………………………………………………

Candidate’s Name:

…………………………………………………………………………………………

Date:

…………………………………………………………………………………………

I hereby declare that preparation and presentation of this [capstone type] were supervised in accordance with the guidelines on supervision of [capstone type] laid down by Ashesi University College.

Supervisor’s Signature:

…………………………………………………………………………………………

Supervisor’s Name:

…………………………………………………………………………………………

Date:

…………………………………………………………………………………………

ACKNOWLEDGEMENT

I would like to express my profound gratitude to all who assisted me in diverse ways to complete this project.

Special gratitude to my supervisor, Mr. Francis Gatsi, for his valuable input without which this project would not have completed successfully.

ABSTRACT

In computer science, practical assignments ensure that students put the theory they learn in class into practice by writing computer programs to solve problems. Practical assignments also play a critical role in assessing students' understanding of course materials. For course facilitators, grading programming assignments is a time-consuming task. The course facilitators must run each student's submission. Moreover, some students copy the code from their friends and change the lexicon and structure. This makes it nearly impossible for the course facilitators to detect plagiarism. A possible solution to these problems is a system that allows course facilitators to write tests that apply automatically to all students' submissions and consequently allocate grades based on test results. To curb the plagiarism issue, the system should have a component that calculates the peer plagiarism index and flags students' submissions that may have plagiarism issues. This applied project is an attempt to develop, test and evaluate such a system. While designing the system, it became apparent that running students' submission and instructors' tests on the server would pose a security threat to the server. After evaluating possible workaround for the issue, we decided to run the submissions and tests on a docker sandbox within a virtual machine. The plagiarism index is calculated by quantifying the lexical and structural similarities. To integrate the two components, we developed an API. To test and demonstrate the workings of the system, we developed a frontend client to consume the critical endpoints of the API. This project is proof of concept that the solution for the problem can be developed and successfully deployed.

TABLE OF CONTENT

Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1 Background context of the Problem

Programming assignments are generally hectic to grade because in most cases the course facilitator must run each student's submission and check for correctness as well as completeness. In some cases, a course facilitator may need to test for good design and documentation. This means that they would need to read each students source code. This can be a daunting and time-consuming job.

The hectic grading process that follows a programming assignment affects the quality and complexity of programming assignments. Course facilitators shy away from assignments that are too complex or too tedious to grade.

According to Coughlin, "Hugely facilitated by computers and the Internet, plagiarism by students threatens the educational quality and professional ethics worldwide though those same technologies can be used to teach correct practices and detect transgressions" [1]. The internet and computers have made code plagiarism an easy task. It is almost impossible for facilitators to detect source code plagiarism among students. When students change the structure and the lexicon of the copied code, often than not it passes as genuine to the facilitator [2].

According to research conducted by Zurich and Dragan in the University of Banjaluka in Bosnia, lexical source code modification includes but not limited to [2] modification of comments, identifiers, variables, and modifiers. Structural code modification includes [2] includes but is not limited to modifying the control structures and loops, changing the order of variables and code blocks, addition of redundant statements, and modification of data structures.

1.2 The significance of the Problem

I conducted a qualitative research to gain insight into why there has been efforts to automate the grading of programming assignments at Ashesi University. The study involved seven interviews with seven faculty which facilitate core computer science courses. All interviewed facilitators acknowledged that code plagiarism was very hard to detect unless it was apparent. Some facilitators feared that the existing auto-grading tools such as VPL and REPL are not flexible enough to provide multiple language support and testing techniques. No facilitator has considered the need for auto-grading and plagiarism detection tools because the status quo has always been manual grading.

Various studies [1][2][3] show that automating the testing process is trivial in theory but complicated in application. In practice, the system must be secure enough for running untrusted student code, scalable to large classes, flexible enough to accommodate different forms of assignments and robust in the face of bugs in the students' programs. If possible, the system should have the ability to grade threaded and distributed programs.

1.3 Proposed Solution

As a solution to the manual grading system and lack of plagiarism detection in programming assignments, a web application is proposed whereby course facilitators can design programming assignments and tests, and students source code solutions to the assignments. The tests are applied to each student's submission and the grade calculated from the test results. The students' submissions are run through a plagiarism checker and their pair-wise plagiarism index calculated. An index above a certain threshold is flagged as plagiarism.

1.3 Related Work

1.3.1 Autograding

A study conducted by David Kay et al. [3] reveals that a system built to grade students programming assignments automatically must be robust and secure in the face of malicious submissions from students. The system must also be scalable to handle large class sizes. A similar study by David Malan [5] shows that security is the most significant threat to such a system. Running code submissions on the server makes the entire system vulnerable. A simple shell program could delete all the files on the server or shut it down entirely. To overcome this challenge, this literature and related literature [6] recommend the use of a constrained sandbox where computer programs can run securely. Examples of such sandboxes include virtual machines and Docker containers. David Malan, a professor at Harvard University, [5] implemented a similar solution for an introductory computer science course at Harvard University. The solution uses Docker containers to create the sandbox. David Malan [5] points out that virtual machines provide more security since they run entirely on a different operating system. The trade-off, however, is the start-up time that virtual machines need every time you need to run a program. Susilo Veri [9], investigated the viability of building a code analyzer and its contributions in improving the teaching and learning process of computer science courses. The analyzer, like code analyzers found in most text editors, would be used by students to reveal logical, lexical and structural bugs without running the code. Her results showed that such a system would save students and instructors time in writing and reviewing source codes. The downside is the analyzer was language specific and could not be used for any language.

1.3.2 Plagiarism Detection

In a technical paper, Georgina Cosmo [7], describes PlaGate, a tool that can be integrated into existing systems to improve performance. According to Georgina, PlaGate also provides graphical evidence of plagiarism which indicates the relative importance of the given source code fragments. This technique of attaching relative importance to different fragments of the source code creates different categories of plagiarisms. In a similar study, Stephen Burrows [8], uses local alignment and lexical similarities to in source codes to detect plagiarism. This method is identical to what is used in detecting plagiarism in other writings. This simplicity of the technique makes it highly scalable to large class sizes as compared to more complicated techniques such as the JPlag and MOSS. Zorac Duric et al. [2] conducted a study into the most common occurrences in source code plagiarism. Zorac Duric et al. [2] found out that all source code plagiarism was either in the form of lexical modification of original source code or structural modification of the source code. Therefore, techniques and algorithms for plagiarism detection must focus on detecting both lexical alteration and structural modification.

Chapter 2: Requirements

2.1 Overview

This section outlines the requirements of the proposed system. The details of the functionalities and intended features of the system are also discussed in depth. The chapter also provides a detailed overview of the functional requirements, functional requirements elicitation, system requirements, external interface requirements, and non-functional requirements.

2.2 Scope

The system intends to replace the manual grading of programming assignments by course facilitators at Ashesi University. The current scope for this system is Ashesi University Computer Science and Engineering departments. The system will serve to reduce time spent by course instructors grading programming assignments. This will allow course instructors more time for other rewarding activities. Ultimately, the system will improve the quality of experience for both instructors and students.

2.3 System Components and Functionalities

The system is divided into various components that are intended to be developed into microservices. Below is a list of the major parts and their anticipated/intended functionalities.

2.3.1 Submittal Component

This component of the system will:

- Allow instructors to set up assignments and create submission slots for various requirements of the assignments,

- Allow students to submit multiple source code as solutions to a programming assignment,
- Allow students to submit ancillary files related to a programming assignment. Such files include but are not limited to documentation files, output results, and students' test,
- Allow students to submit the assignment multiple times as long as the deadline is not passed. The submission timestamp of the last submission is recorded in case of late submission,
- Compile the submitted source files, and
- Run compiled source code against the published test cases.

2.3.2 Grading Component

This component of the system will:

- Allow course instructors to write tests and provide test cases for various assignments,
- Run the compiled student source code against the tests specified by the course instructor(s), and
- Collect the results of each student tests and publish them to the database.

It is important to note that the system mainly checks for correctness and completeness and may not be able to assign grades on sound design and documentation.

2.3.3 Plagiarism Check Component

This component of the system will:

- Generate an intermediate representation of the submitted source codes,
- Look for plagiarized material in the source code and compare similarity, and

- Alert the course instructor in case there is any two source codes are flagged as similar.

A plagiarism index above the threshold may not always indicate a case of plagiarism; the facilitator will have to go through similar source codes and make a judgment call.

2.4 User Roles and Responsibilities

This section describes how different stakeholders will interact with the system and what kind of activities the system is required to support:

2.4.1 Course Instructors

Course instructors will use the system to create new assignments and open submission slots for all submission requirements. Submission requirements may include one or multiple source codes, documentation, and output results. For each requirement, the instructor will create a different submission slot. The instructor will also write tests that will be applied to the student's submissions. An instructor may write a test for the source codes to determine whether they display the intended behavior. An instructor may also write tests to be applied to the students' output results to check whether the output results are as intended. The system will help automate the check for correctness and plagiarism, but it does not check for good design and good documentation. The lecturer may need to through the students' source code to check for good design and documentation.

2.4.2 Students

Students will use the system to submit programming assignments. A student may submit multiple times given that the submission deadline has not passed. Students will see the results of the published test cases soon after they present. The results of the rest of the test cases will appear only after the submission deadline.

2.5 Requirements Gathering

To gather requirements a mixed research approach that involved both qualitative and quantitative research was explored. Interviews, observation, and emersion were the main tools employed. Emersion included job shadowing a programming course facilitator at Ashesi University. The system stakeholders, i.e., students, lecturers, and school administrators were the correspondents of the research. The information sort after these stakeholders includes but is not limited to:

- How does a course facilitator grade a typical programming assignment?
- How much time does it take a facilitator to grade a typical programming assignment?
- Does the facilitator detect plagiarism in programming assignments?
- How does the facilitator detect plagiarism in programming assignments?
- How does a facilitator track development and progress for each student?
- How do students submit programming assignments?
- Do students get feedback from their facilitator after a programming assignment?
- What Integrated Development Environments (IDE) do most students prefer?
- What tools are available for automatic code submission?
- Why has Ashesi not adopted any of the available tools?
- What kind of data will this system be dealing?

2.6 Requirements Analysis

Analyzing input from the stakeholders, and notes from observation and emersion revealed the following needs:

- Facilitators should invite students to course,
- An administrator should invite lecturer to register in the system,
- Only Ashesi email format can register in the system,
- Facilitators should create courses,
- Courses facilitator should create assignments,
- Students should view assignments,
- The system should check for correctness and completeness,
- The system should check for good design and documentation,
- The system should be safe to run students untrusted code,
- The system should support multithreaded applications,
- Students should be able to submit multiple source files,
- Students should be able to submit ancillary files alongside the source code,
- Students should be able to submit multiple times before the deadline,
- There should be a real-time scoreboard that shows students' scores under their pseudocodes,
- The system should be scalable to large classes,
- The system should be fast enough and reliable,
- The system should be able to detect both lexical and structural plagiarism,
- The system should be able to report on student progress,
- The student should be pluggable into school management systems.

2.7 Requirements Organization and Specification

In organizing the requirements from the research participants, we identified themes and classified all their needs into the following categories:

Table 1: 2.7.1 Requirements in categories

Submission	**Plagiarism Checker**	**Auto-grading**	**Authentication and Sessions**
✓ Submit multiple source files ✓ Submit ancillary files ✓ Submit multiple times before the deadline	✓ Scalable for a large class ✓ Fast and reliable ✓ Detect lexical and structural plagiarism	✓ Check for correctness and completeness ✓ Check for good design and documentation ✓ Run students' untrusted code ✓ Support multithreaded programs ✓ Real-time scoreboard ✓ Scalable for a large class ✓ Fast and reliable	✓ Admin resisters facilitators ✓ Facilitators create courses ✓ Facilitators create assignments in courses ✓ Only Ashesi email format is accepted ✓ Students can view all courses ✓ Students can view all assignments in course

Table 2: 2.7.1 Requirements in categories

Students	✓ **Course Facilitators**	**Admin**	**System**
✓ Can view all courses they are registered in ✓ Can view all assignments in courses they are registered in	✓ Can create courses ✓ Can create assignments and write tests ✓ Can view all students progress	✓ Can invite facilitators to register ✓	✓ Scalable ✓ Fast ✓ Reliable ✓ Robust ✓ Flexible ✓ Pluggable on other systems

✓ Should be able to submit multiple source code files ✓ Should be able to submit ancillary files ✓ Should see their progress ✓ See their results on the live scoreboard	✓ Can invite students to register		

2.7.1 Functional Requirements

Functional requirements for an administrator include:

- Login/Register: The administrator should have secure access to their account. An email and password combination would be required to grant the administrator access. In case the administrator forgets their password, they can recover it through their emails.
- Faculty invites: The administrator can send email invites to new faculties to join the platform.
- Approve course creation: When a faculty creates a course, the status of the course will be pending, and no student can join the course unless the facilitator approves it.
- Approve students course registration: When students join a course their join status will be pending until the administrator approves them.
- Unregister faculty: The administrator can unregister a faculty from the platform in case the faculty is no longer with the institution.
- Unregister student: The administrator can unregister a student from the platform in case the student is no longer with the student.

- Access to all courses' records: The administrator has view access to all courses' records.
- Access to all students' records: The administrator has view access to all the students' records.

Functional requirements for a course facilitator include:

- Login/Register: The course should have secure access to their account. An email and password combination would be required to grant them access. In case the facilitator forgets their password, they can recover it through their emails.
- Create a course: The course facilitator has the right to create a new course on the platform with the approval of the administrator.
- Create an assignment: The course facilitator can create a new assignment in the course that they facilitate.
- Write tests: For an assignment, the course facilitator can write tests that will be applied to the students' submissions.
- Receive plagiarism notifications: After the deadline, the system computes the peer plagiarism index for every students' submission. If for a student, the plagiarism index is above a certain threshold, the system notifies the course facilitator.
- View students' plagiarism index: The course facilitator has view access to the computed plagiarism index of all the students registered to a course.

Functional requirements for a student include:

- Login/Register: The student should have secure access to their account. An email and password combination would be required to grand the student's access. In case the student forgets their password, they can recover it through their emails.
- Register in a course: Students can register to a course with the approval of the administrator.

- Upload assignments: A student can submit source code files for an assignment multiples times before the deadline. Submission after the deadline is accepted but the submission timestamp will be recorded.
- View Assignments score: Students can view their scores after an assignment's deadline.
- Lodge complain: If a student suspects that their assignment grade is faculty, they can submit a complaint that will be viewed by the course's facilitator.
- View assignments plagiarism index: Students can view their plagiarism index after an assignment's submission deadline.
- View course progress: Students can view a graph of their grades.

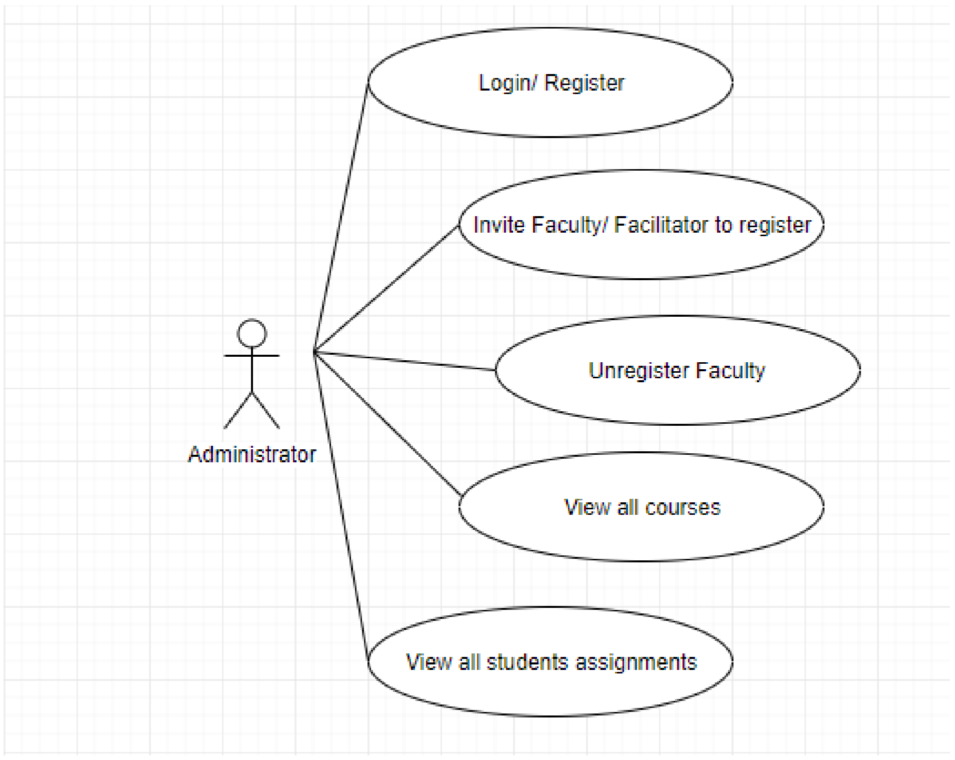
2.7.2 Non-Functional Requirements

- Flexibility: The system must be flexible enough to accommodate different forms of assignments
- Performance: The functional components of the system must run in a reasonable amount of time
- Robustness: The system should be robust in the face of bugs in the students' source codes
- Scalability: The system should be able to accommodate a class size of up to 1000 students without breaking.
- Security: The system should be able to run code in isolated safe mode in case of malicious source codes

2.8 Users

2.8.1 Admin user class

Use case:



Login/ Register
Invite Faculty/ Facilitator to register
Administrator
Unregister Faculty
View all courses
View all students assignments

Figure 1: 2.8.1 Use case for the admin user class

Scenario:

Head of the computer science department at Ashesi University wants computer sciences faculties to use the system for programming assignments. To do this, the head of computer science departments will register an administrator account and invite other faculties to join the system through emails. If a faculty leaves the school the head of the computer science department will unregister them from the platform. The administrator can view all courses on the platform, and all students progress reports.

2.8.2 Instructor user class

Use case:

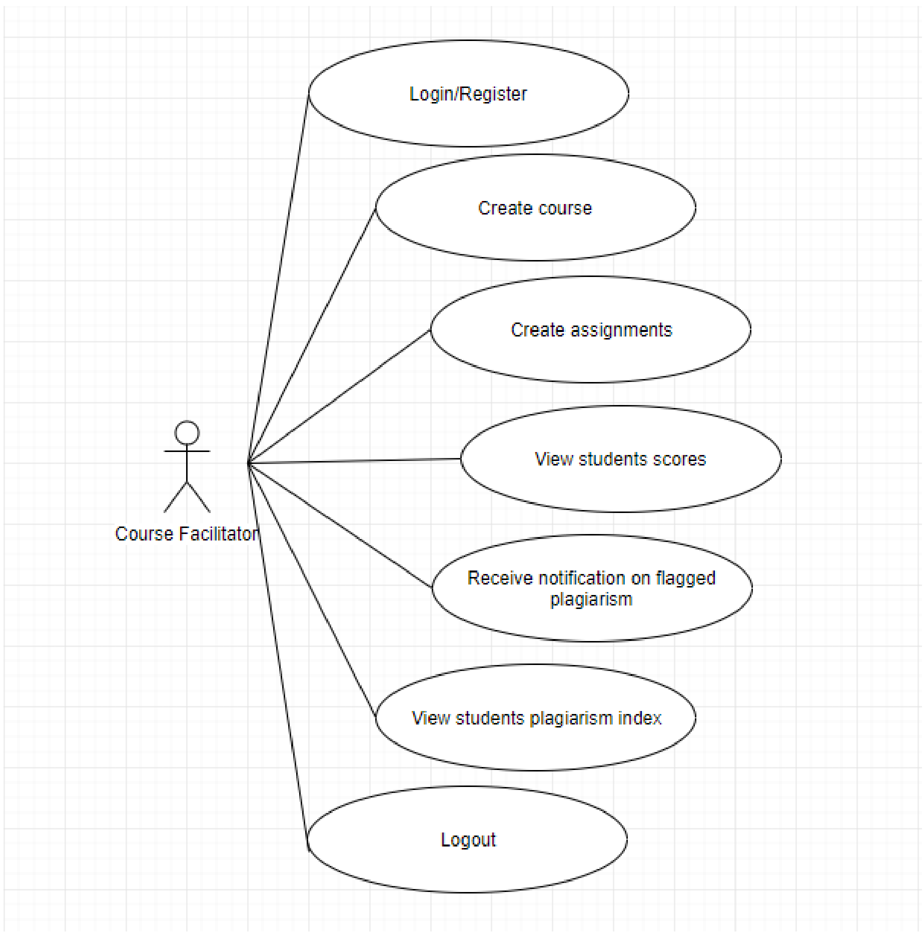


Figure 2: 2.8.2 Use case for the instructor user class

<u>Scenario:</u>

A data structures lecturer at Ashesi University gets an email invite from the Head of Department to register on the platform. After registration, he/she will create the Data Structures course on the platform. The lecturer will create an assignment and write tests that will be applied to students' submissions. After the assignment deadline, the facilitator will view the assignment scores. The lecturer will also receive notifications in case any student's code has been flagged as plagiarised.

2.8.3 Student user class

<u>Use Case:</u>

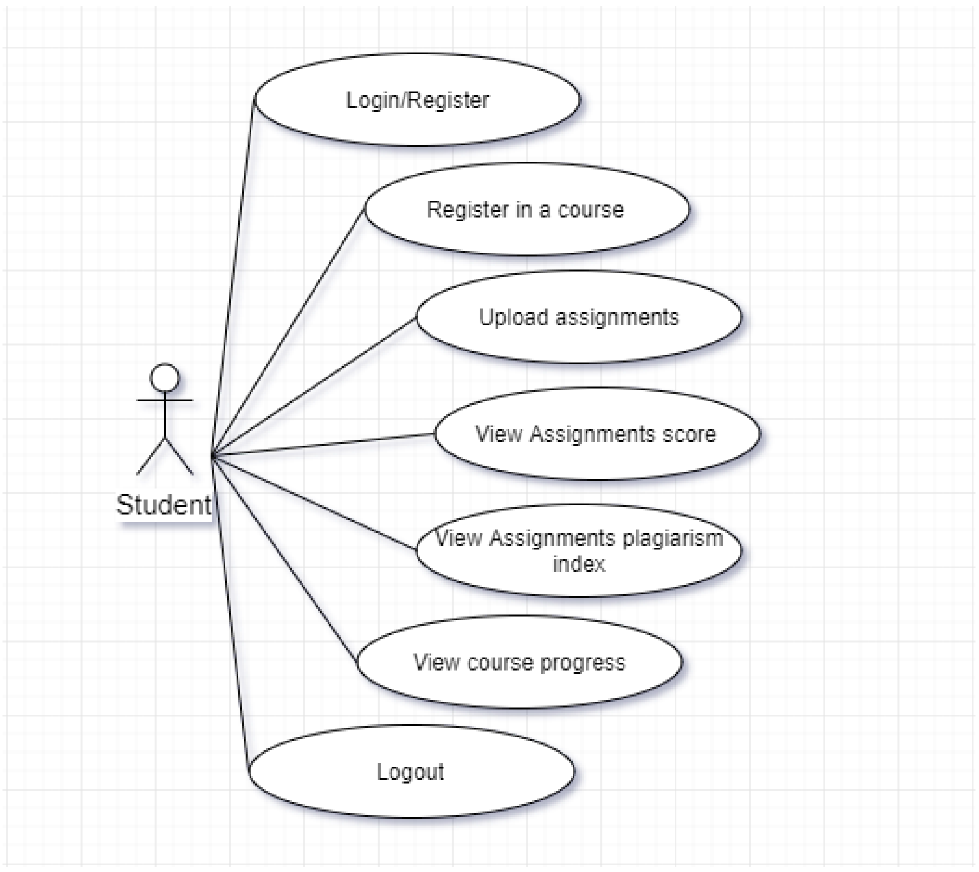


Figure 3: 2.8.3 Use case for the student user class

<u>Scenario:</u>

A student registered on the platform through the Ashesi email will be able to register to the Data Structures course. The students will see all due assignments and be able to submit the course code files and ancillary files multiple times before the deadline. After the deadline, the student will get to view the assignments score and plagiarism index. The student will also view a graph of his scores.

Chapter 3: Architecture and Design

3.1 Overview

This section provides a summary of the architecture and design of the proposed application. A high-level design which satisfies the requirements specified earlier will be provided in this section.

3.2 System Overview

The web application has the following components:

- Three users (administrator, course facilitators, and student),
- A front-end client that allows each user to have a different view
- Authentication service that allows each user to access their account
- A plagiarism detection service that checks the plagiarism index for each student’s submissions
- An auto-grading service that applies the facilitators' tests on the students' submissions and computes the scores
- Database for the persistence of data in the system
- File system for storing students submissions

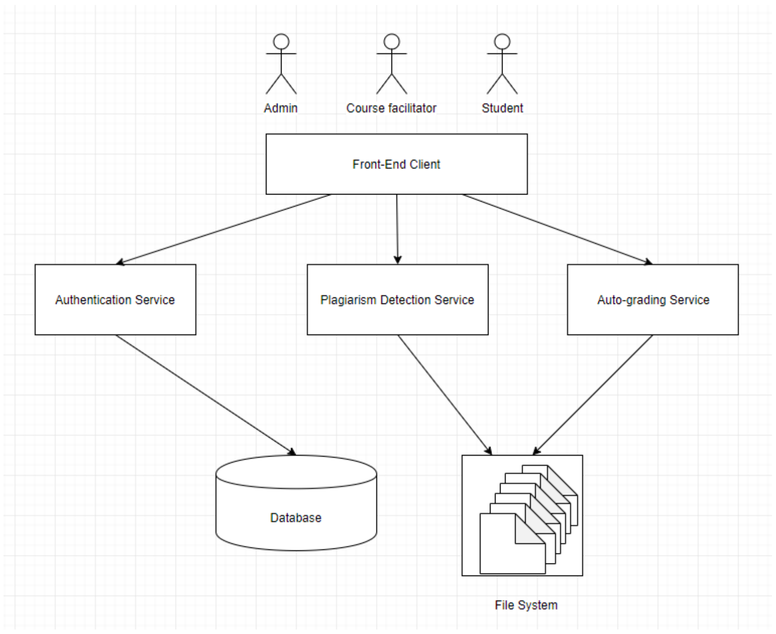


Figure 4: 3.2 High-level system architecture

3.3 System Architecture

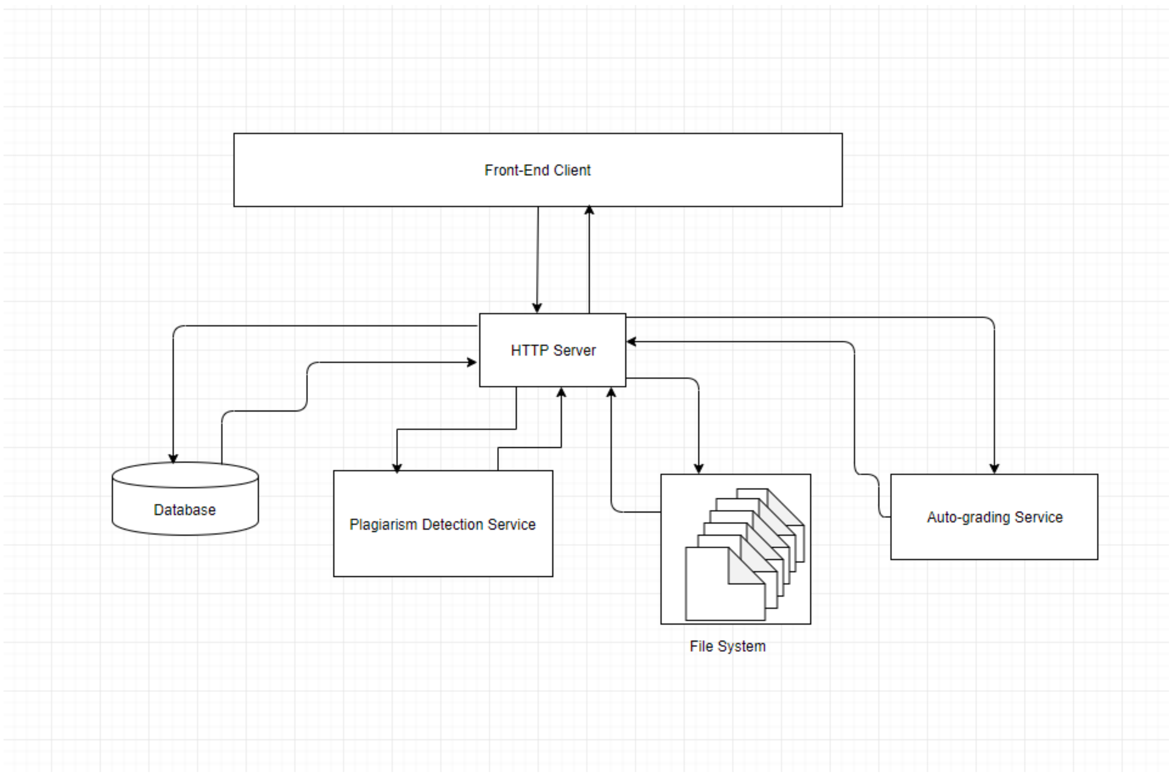


Figure 5: 3.3 System architecture

3.3.1 Front-end Client

The front-end client of this application will consist of the home page and views for each user. The purpose of the front-end client is to consume a few critical endpoints of the API just to allow for integration testing and demonstration of the working of the system.

3.3.2 HTTP Server (API)

Although the application architecture is Micro-Services Architecture, the HTTP server acts as the controller and the link between the front-end client and all the services. The HTTP server will consist of the following services:

- login user: This service will authenticate the users and redirect them to their dashboards
- register user: This service will register new users and redirect them to their dashboards
- update user: This service will update user details
- delete user: This service will delete a user from the system
- create course: This service will create a new course on the platform
- update course: This service will update course details
- delete course: This service will delete the course from the system
- join course: This service will add a student into a course
- leave course: This service will remove a student from a course
- submit file: This service will upload a file into the file system
- delete file: This service will delete a file from the file system
- test: This service will apply the facilitators test to the students' submissions
- check plagiarism: This service will run the plagiarism checker on all student submissions

- create complain: This service will create a new complain
- delete complain: This service will delete a complain
- close complain: This service will close a complain
- create assignment: This service will create a new assignment
- delete assignment: This service will delete the assignment
- submit assignment: This service will submit call ‘submit file’ service in order to upload the assignment files

3.3.3 Database

The database will contain the following primary entities:

- Admin: This entity will store all registered admins’ details
- Faculty: This entity will store all registered faculties’ details
- Student: This entity will store all registered students’ details
- Course: This entity will store all registered courses’ details
- Assignment: This entity will store all registered assignments’ details

The database will contain the following secondary entities:

- Assignment Submission: This entity will store all assignment submissions’ details
- Course Enrolment: This entity will store all courses’ enrolment details
- Complaints: This entity will store all complaints lodged by students

3.3.4 File System

The filesystem will store all the files uploaded into the system.

3.3.5 Plagiarism Detection Service

The plagiarism checker will determine whether students have plagiarised among each other and return each students plagiarism index.

3.3.6 Auto-grading service

The auto grading service will apply the tests to each student’s source code and return the results for each student in a JSON format. The auto-grading service will run in an isolated environment to provide protection against malicious programs.

3.4 Component Diagram

The diagram below shows the main components of the system and the relationship between them.

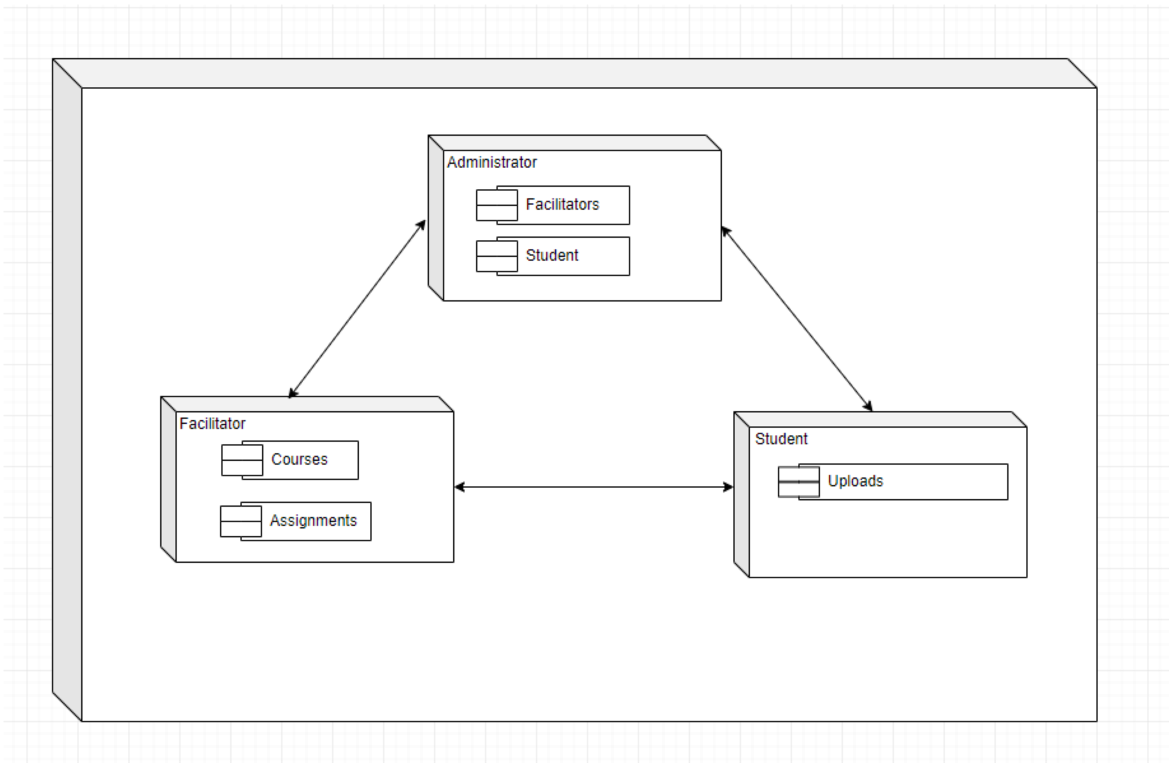


Figure 6: 3.4 Component diagram

3.5 Activity Diagrams

3.5.1 Admin

An administrator will require to be authenticated to access their accounts. If the admin has an account, they will log in into the system; otherwise, they will need to be registered through an existing admin. Once authenticated, the admin can send an invite to an unregistered faculty. If any faculty has created a course, the admin may decide to approve it or decline the course creation. If there is a faculty that has recently resigned the admin can unregister them from the system.

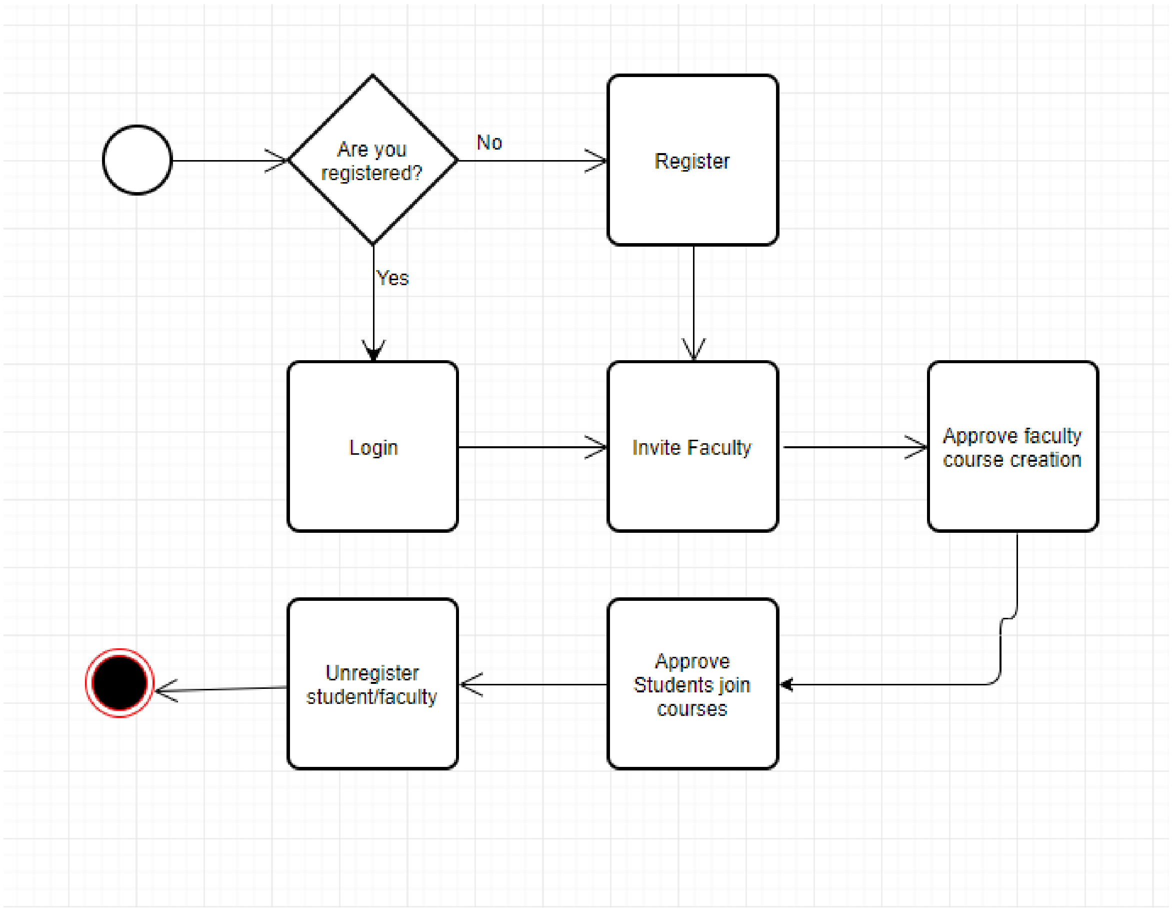

Are you registered?
No
Register
Yes
Login
Invite Faculty
Approve faculty course creation
Unregister student/faculty
Approve Students join courses

Figure 7: 3.5.1 Admin activity diagram

3.5.2 Course Facilitator

A course facilitator will require to be authenticated to access their account. If the faculty is not registered, they would need to ask the admin to send them an invite. Once authenticated, a facilitator can create a new course and wait for the admin to approve it. Once the course creation is approved, the facilitator can create an assignment within the course. All students who have joined the course will view the assignment and submit the source code as well as the ancillary files. After the deadline, the tests will be applied to the source codes and results computed. The faculty can view students results. The plagiarism indices will also be computed after the deadline and the results stored on the database. The faculty can also view the results. The faculty will receive notifications for cases that are flagged as plagiarism.

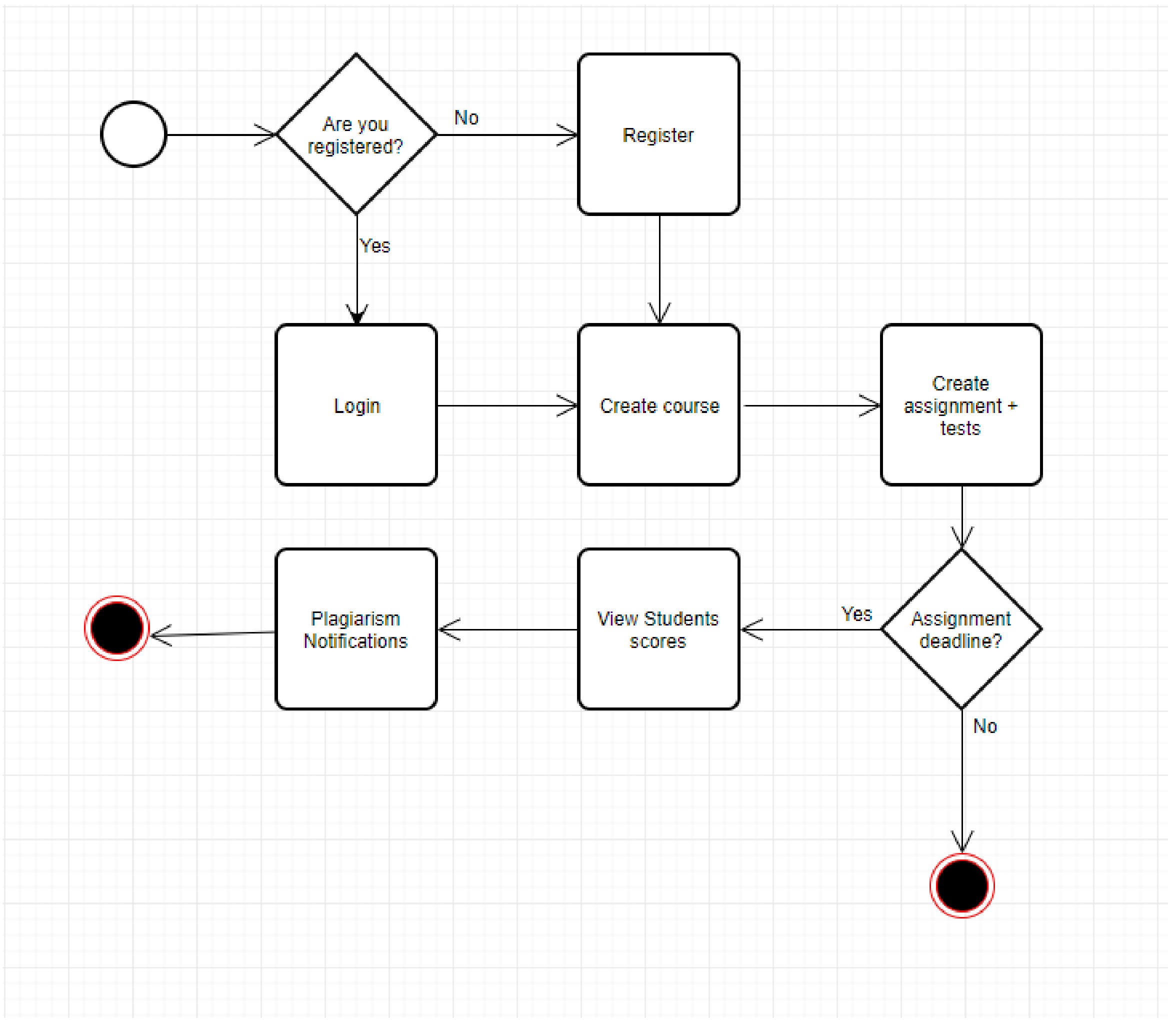


Figure 8: 3.5.2 Instructor activity diagram

3.5.3 Student

An authenticated student can join a course. After the administrator has approved the join request, the student can view and submit assignments within the course. After the assignment deadline, the scores and the plagiarism indices will be computed by the auto-grader and plagiarism checker respectively. The students may view the results for both. The student can also view their grades graphically.

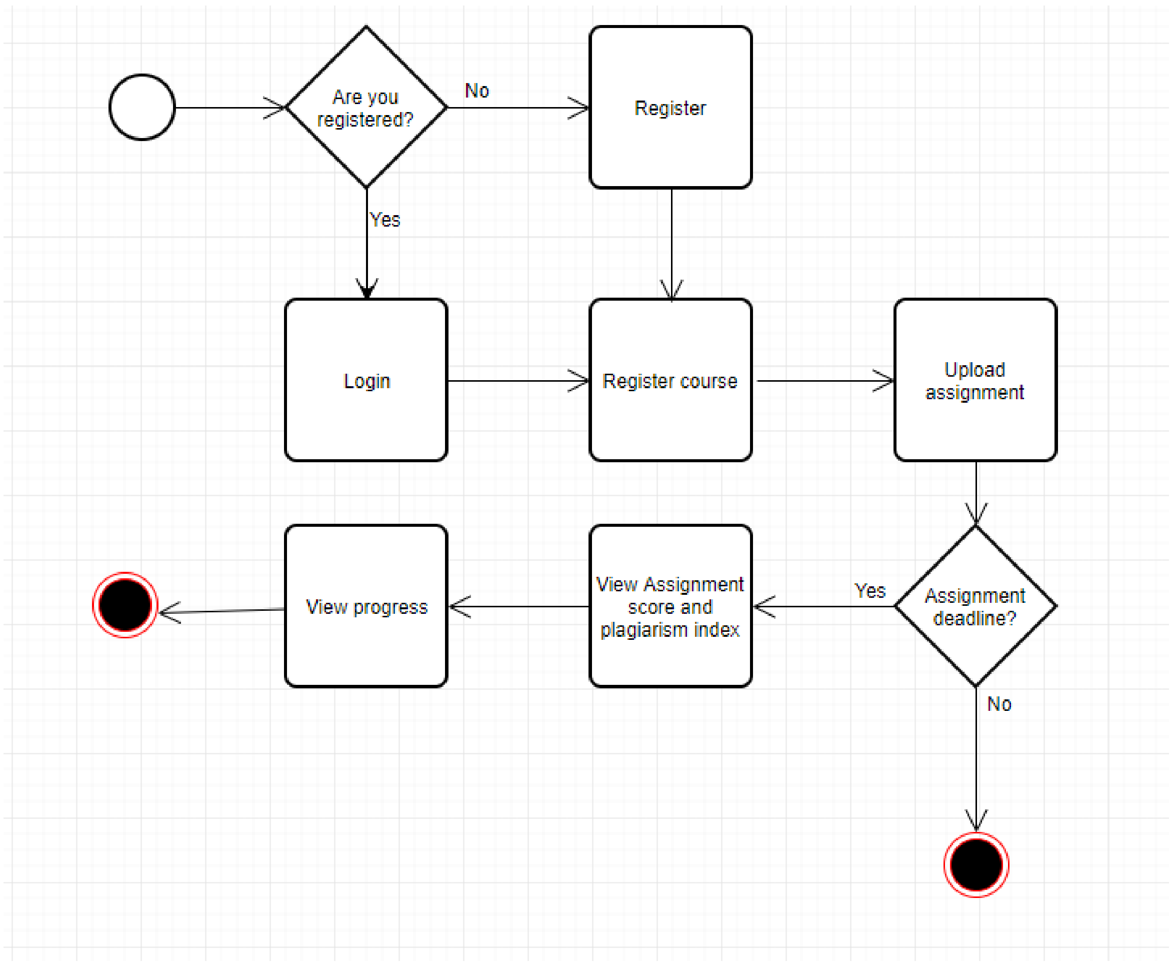


Figure 9: 3.5.3 Student activity diagram

3.6 Extended Entity Relational Diagram

The diagram below shows the relationships of all entities in the system and their relationships.

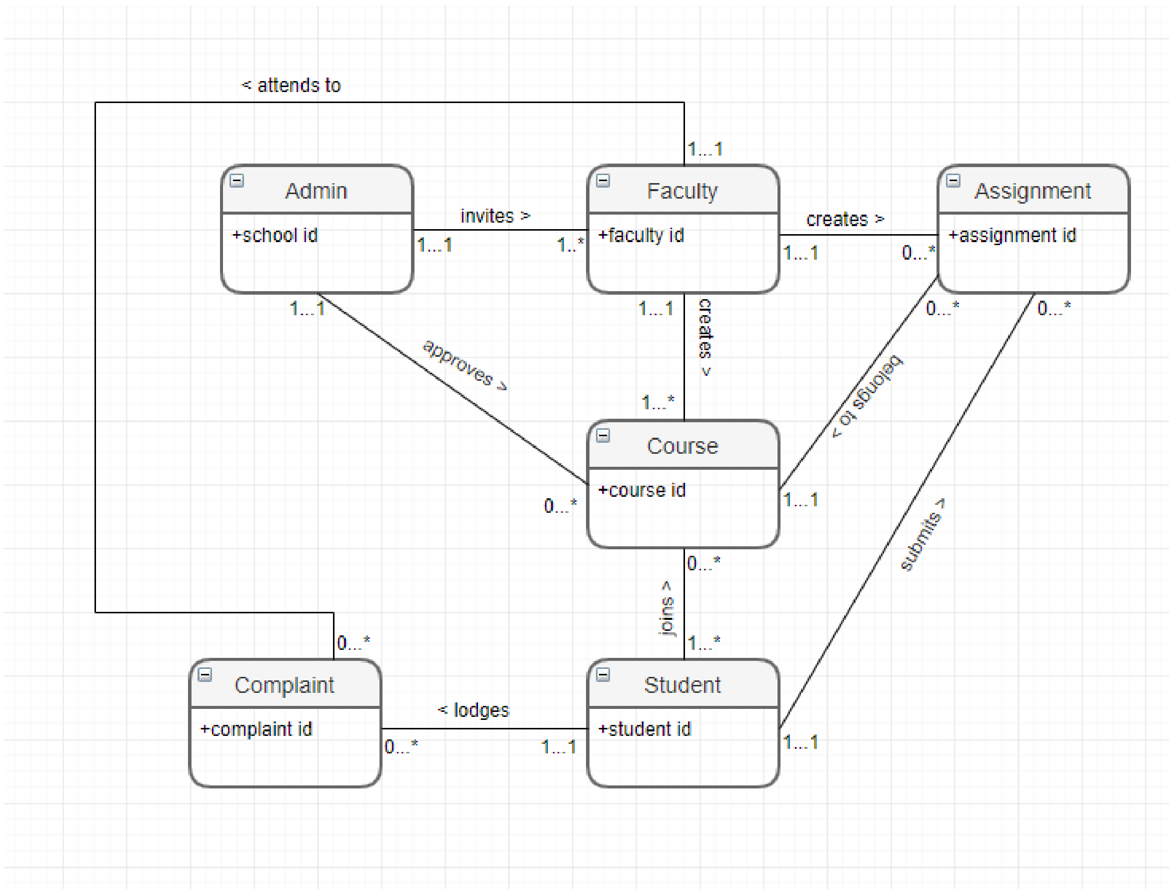


Figure 10: 3.6 Extended Entity Related Diagram

3.7 Database Architecture

Below is a description of the database tables and their relationships:

- **Admin:** The admin table has the *adminid* as its primary key.
- **Student:** The student table has the *studentid* as its primary key. The *studentid* is a foreign key in the Complaint, Assignment Submission, and Course Enrolment tables.
- **Faculty:** The faculty table has the *facultyid* as its primary key. The *facultyid* is a foreign key in the Complaint table.
- **Course:** The course table has a *courseid* as its primary key. The *courseid* is a foreign key in the Assignments table.
- **Assignment:** The assignment table has *assignmentid* as its primary key.

- **Complaint:** The complaint table has *complaintid* as its primary key.
- **Assignment submission:** The assignment submission table has the *submissionid* as its primary key.
- **Course Enrolment:** The course enrolment table has the *enrolmentid* as its primary key.

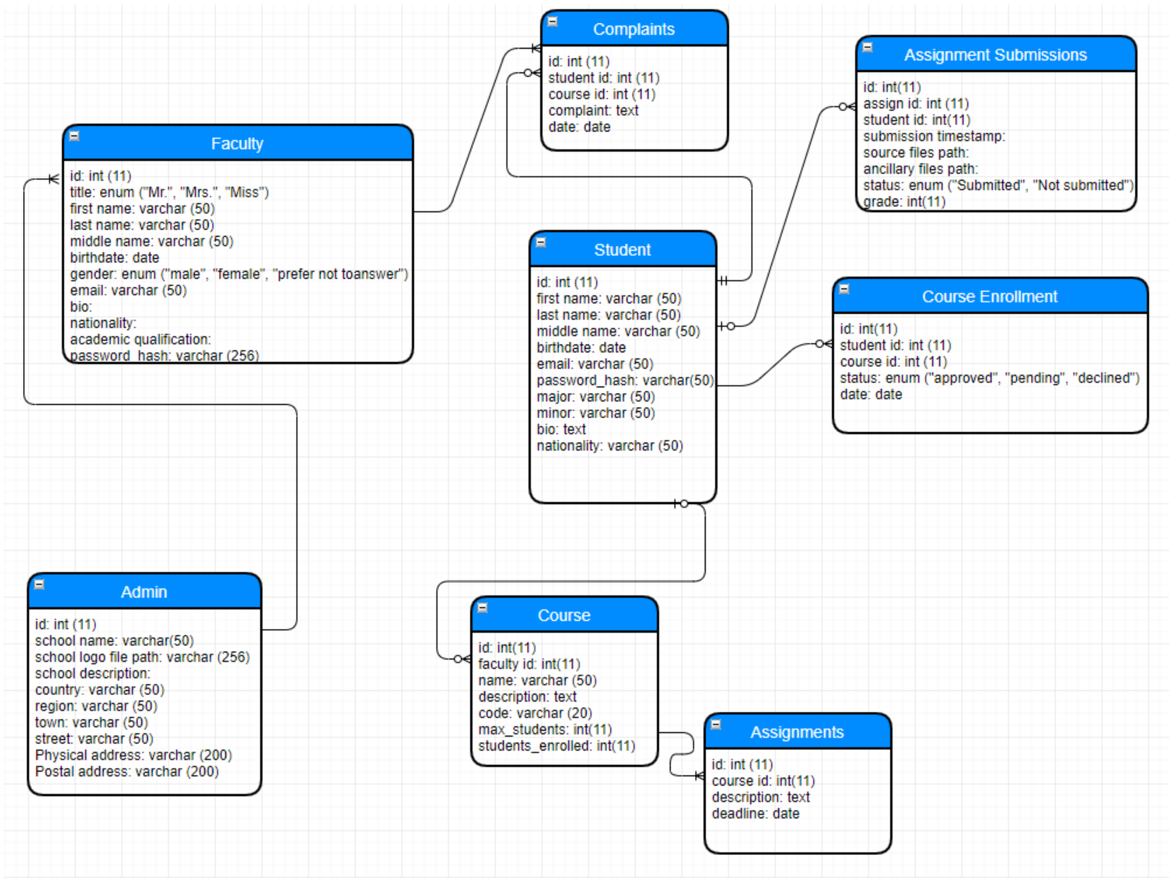


Figure 11: 3.7 database architecture

3.8 File System

A lot of the system data involves files that will be stored in the file system as designed below. The directory and files naming convections are as displayed on the diagram below.

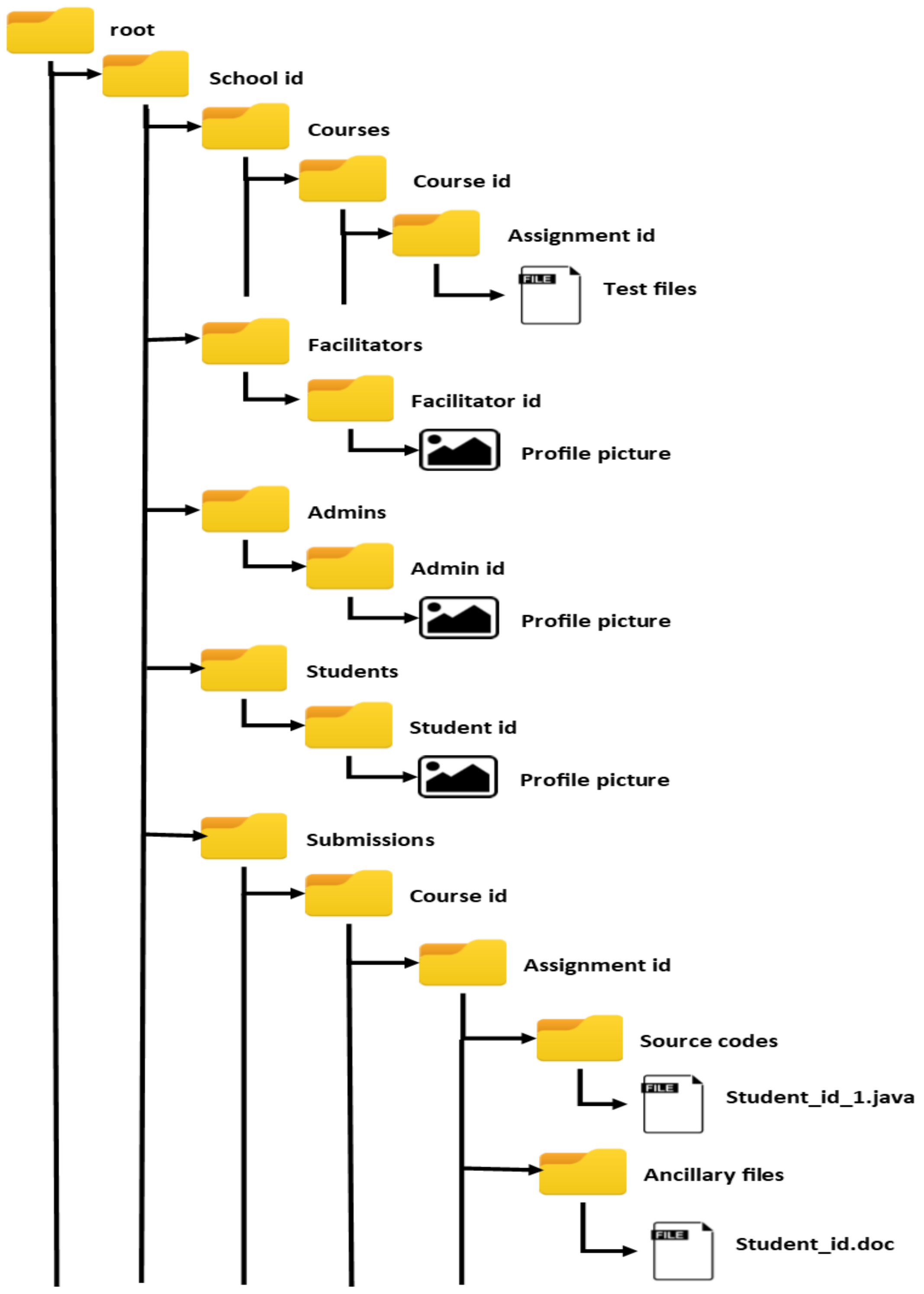


Figure 12: 3.8 File System

Chapter 4: Implementation

4.1 Overview

This section describes the tools and technologies used in the development of the system. It also describes the implementation techniques and algorithms used in the implementation of the system.

4.2 Tools

4.2.1 JavaScript

It is an object-oriented programming language mostly used to render interactive web applications. Recently, JavaScript has gained prominence as a server-side scripting language. In this project, JavaScript is used for both services. It is used in the backend because it allows for asynchronous IO.

4.2.2 Node.js

Node.js is an open source, cross-platform JavaScript run-time environment that executes JavaScript code outside the browser. It is used in this project because it allows the entire application to be written in a single language, JavaScript.

4.2.3 HTML

HTML is an acronym that stands for Hyper Text Mark-up Language. HTML is used in this project to design and create the frontend client.

4.2.4 Shell Script

Shell Script is a program designed to be run by the Unix shell. Shell scripts are used in this project as wrappers, to set up the docker sandbox and run student source code on the docker containers.

4.3 Libraries

4.3.1 Bcrypt

Bcrypt is a library used to hash passwords based on the Blowfish cipher. Bcrypt is available in different languages but this project employs the npm bcrypt library. This library is used to hash passwords and all sensitive information.

4.3.2 Nodemon

Nodemon is a npm library used to monitor the application script during development. It allows for live changes in the application without necessarily having to restart the server. Nodemon is used in the development of this project.

4.3.3 Vision

Vision is a npm library for template rendering plugin support for hapi.js, a Node.js framework. Vision enables applications developed using hapi.js framework to render dynamic templates and dynamic contexts and helpers.

4.3.4 Apollo-Server-Hapi

This library integrates the Apollo Server into the hapi.js framework. The Apollo server is an open-source GraphQL server.

4.3.5 Hapi-Swagger

This is a plugin library for the hapi.js framework that is used to document the API interface in a project. This library is used in this project to document the database API interface.

4.3.6 Mongoose

Mongoose is a MongoDB object modelling library designed to work in an asynchronous environment. It provides built-in typecasting, validation, query building, and business logic hooks. It is used in this project to connect to an online MongoDB database, validate data before insertion and build and execute queries.

4.4 Frameworks

4.4.1 MongoDB

MongoDB is a cross-platform document-oriented database. MongoDB was used in this project as the main database technology.

4.4.2 Hapi.Js

Hapi.js is a framework for building applications and services using JavaScript as the server-side scripting language and Node.Js as the run-time environment. Hapi.js focusses on writing reusable application logic.

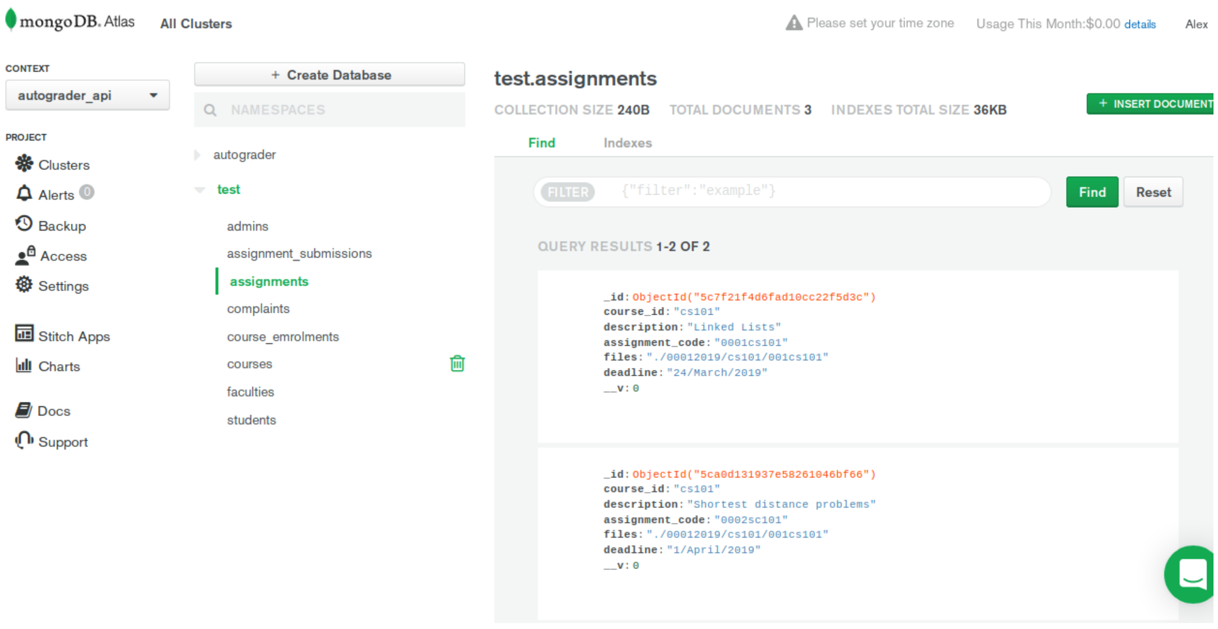
4.5 Description of components

The most significant components and sub-components to the general functionality of the system are described and analyzed below.

4.5.1 Database API

This component handles all the processes that involve accessing information from the database or making insertions or deletion on the database. The frontend client makes API calls in the form of HTTP and https requests to this component to access the database.

Below is an image of the online database.

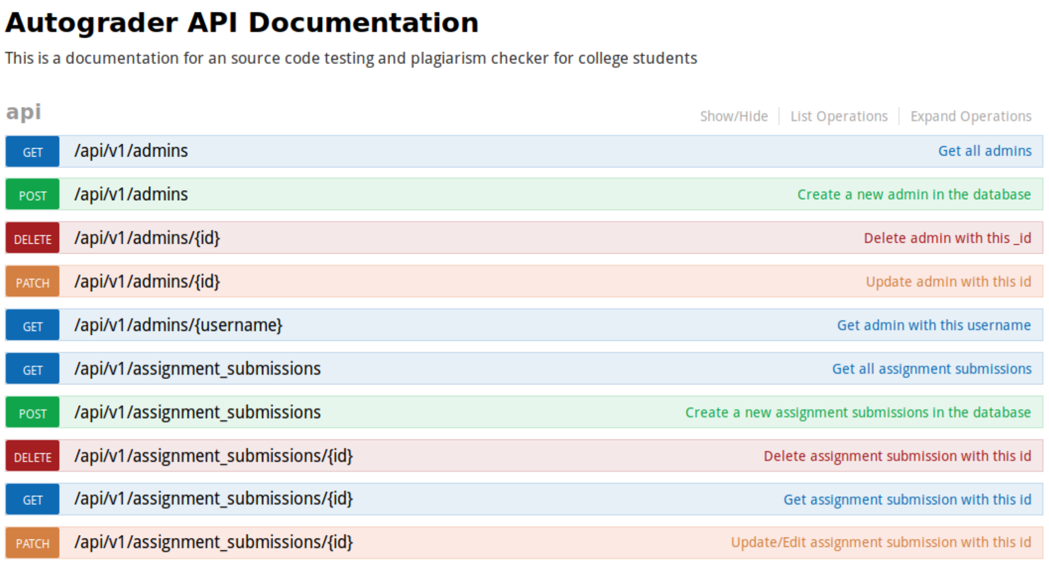


```
_id: ObjectId("5c7f21f4d6fad10cc22f5d3c")
course_id: "cs101"
description: "Linked Lists"
assignment_code: "0001cs101"
files: "./00012019/cs101/001cs101"
deadline: "24/March/2019"
__v: 0
```

```
_id: ObjectId("5ca0d131937e58261046bf66")
course_id: "cs101"
description: "Shortest distance problems"
assignment_code: "0002sc101"
files: "./00012019/cs101/001cs101"
deadline: "1/April/2019"
__v: 0
```

Figure 13: 4.5.1 MongoDB Atlas online portal

Below are illustrations of the Database API documentation.



Autograder API Documentation

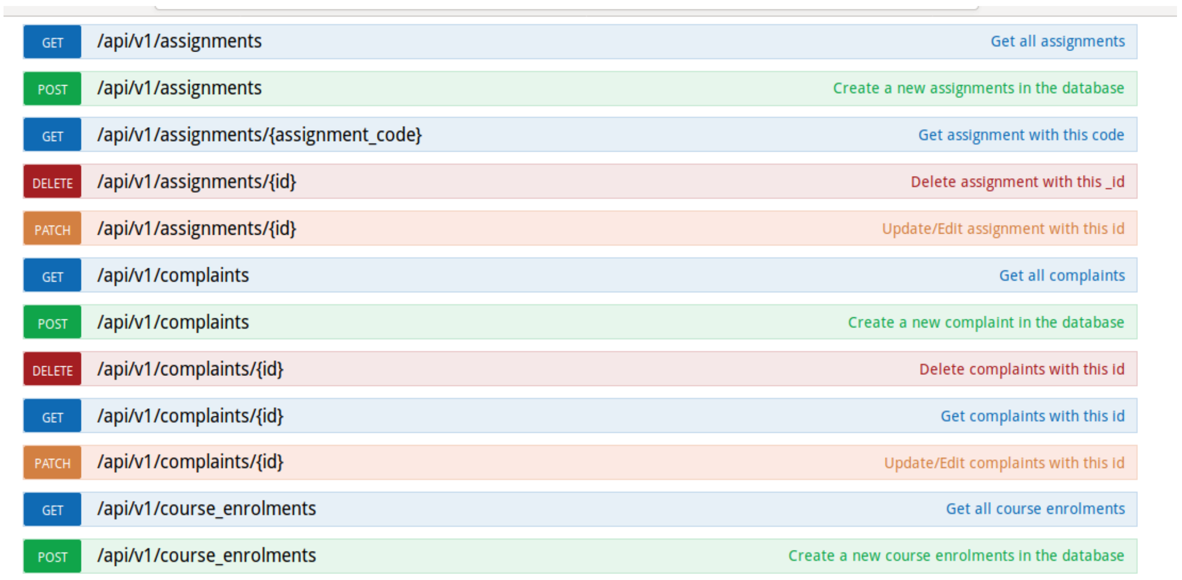
This is a documentation for an source code testing and plagiarism checker for college students

api

Show/Hide | List Operations | Expand Operations

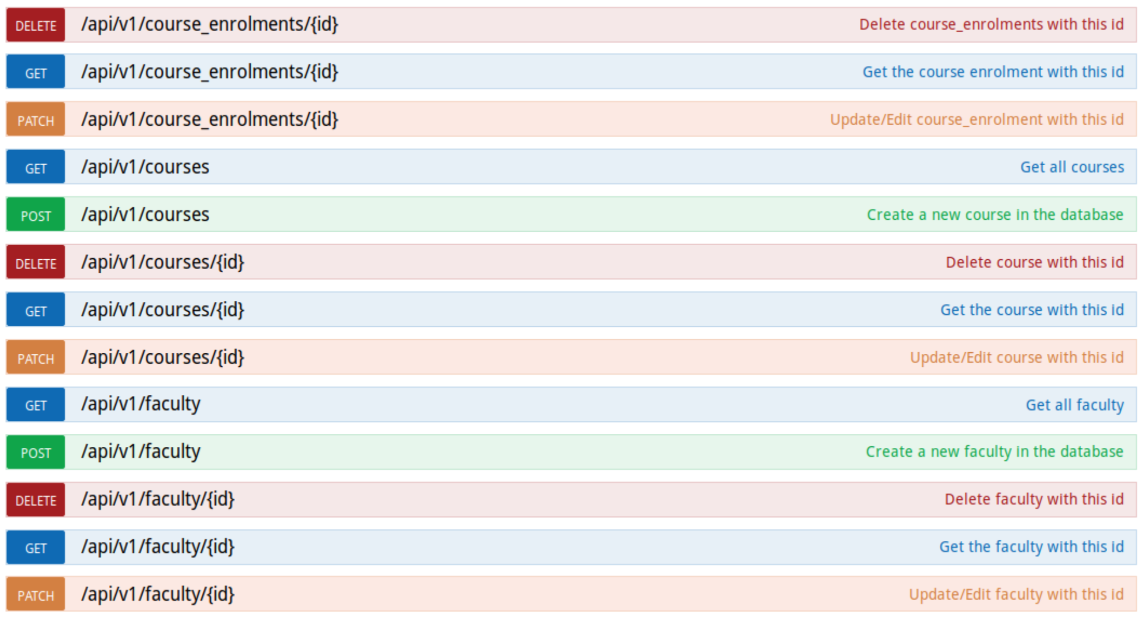
GET	/api/v1/admins	Get all admins
POST	/api/v1/admins	Create a new admin in the database
DELETE	/api/v1/admins/{id}	Delete admin with this _id
PATCH	/api/v1/admins/{id}	Update admin with this id
GET	/api/v1/admins/{username}	Get admin with this username
GET	/api/v1/assignment_submissions	Get all assignment submissions
POST	/api/v1/assignment_submissions	Create a new assignment submissions in the database
DELETE	/api/v1/assignment_submissions/{id}	Delete assignment submission with this id
GET	/api/v1/assignment_submissions/{id}	Get assignment submission with this id
PATCH	/api/v1/assignment_submissions/{id}	Update/Edit assignment submission with this id

Figure 14: 4.5.1 API documentation



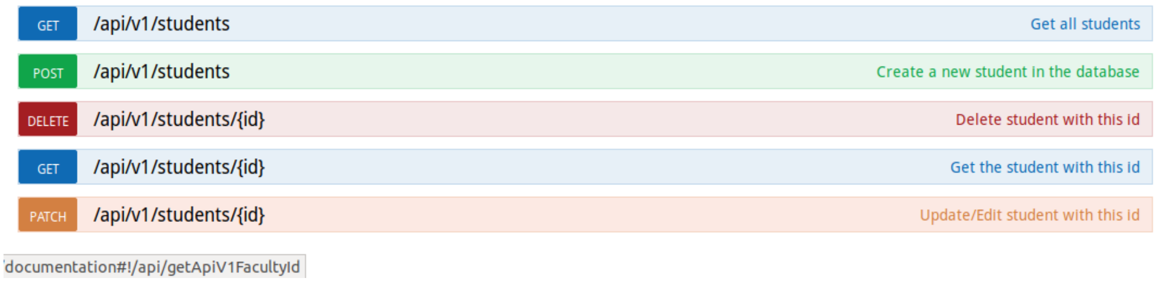
GET	/api/v1/assignments	Get all assignments
POST	/api/v1/assignments	Create a new assignments in the database
GET	/api/v1/assignments/{assignment_code}	Get assignment with this code
DELETE	/api/v1/assignments/{id}	Delete assignment with this _id
PATCH	/api/v1/assignments/{id}	Update/Edit assignment with this id
GET	/api/v1/complaints	Get all complaints
POST	/api/v1/complaints	Create a new complaint in the database
DELETE	/api/v1/complaints/{id}	Delete complaints with this id
GET	/api/v1/complaints/{id}	Get complaints with this id
PATCH	/api/v1/complaints/{id}	Update/Edit complaints with this id
GET	/api/v1/course_enrolments	Get all course enrolments
POST	/api/v1/course_enrolments	Create a new course enrolments in the database

Figure 15: 4.5.1 API documentation



DELETE	/api/v1/course_enrolments/{id}	Delete course_enrolments with this id
GET	/api/v1/course_enrolments/{id}	Get the course enrolment with this id
PATCH	/api/v1/course_enrolments/{id}	Update/Edit course_enrolment with this id
GET	/api/v1/courses	Get all courses
POST	/api/v1/courses	Create a new course in the database
DELETE	/api/v1/courses/{id}	Delete course with this id
GET	/api/v1/courses/{id}	Get the course with this id
PATCH	/api/v1/courses/{id}	Update/Edit course with this id
GET	/api/v1/faculty	Get all faculty
POST	/api/v1/faculty	Create a new faculty in the database
DELETE	/api/v1/faculty/{id}	Delete faculty with this id
GET	/api/v1/faculty/{id}	Get the faculty with this id
PATCH	/api/v1/faculty/{id}	Update/Edit faculty with this id

Figure 16: 4.5.1 API documentation

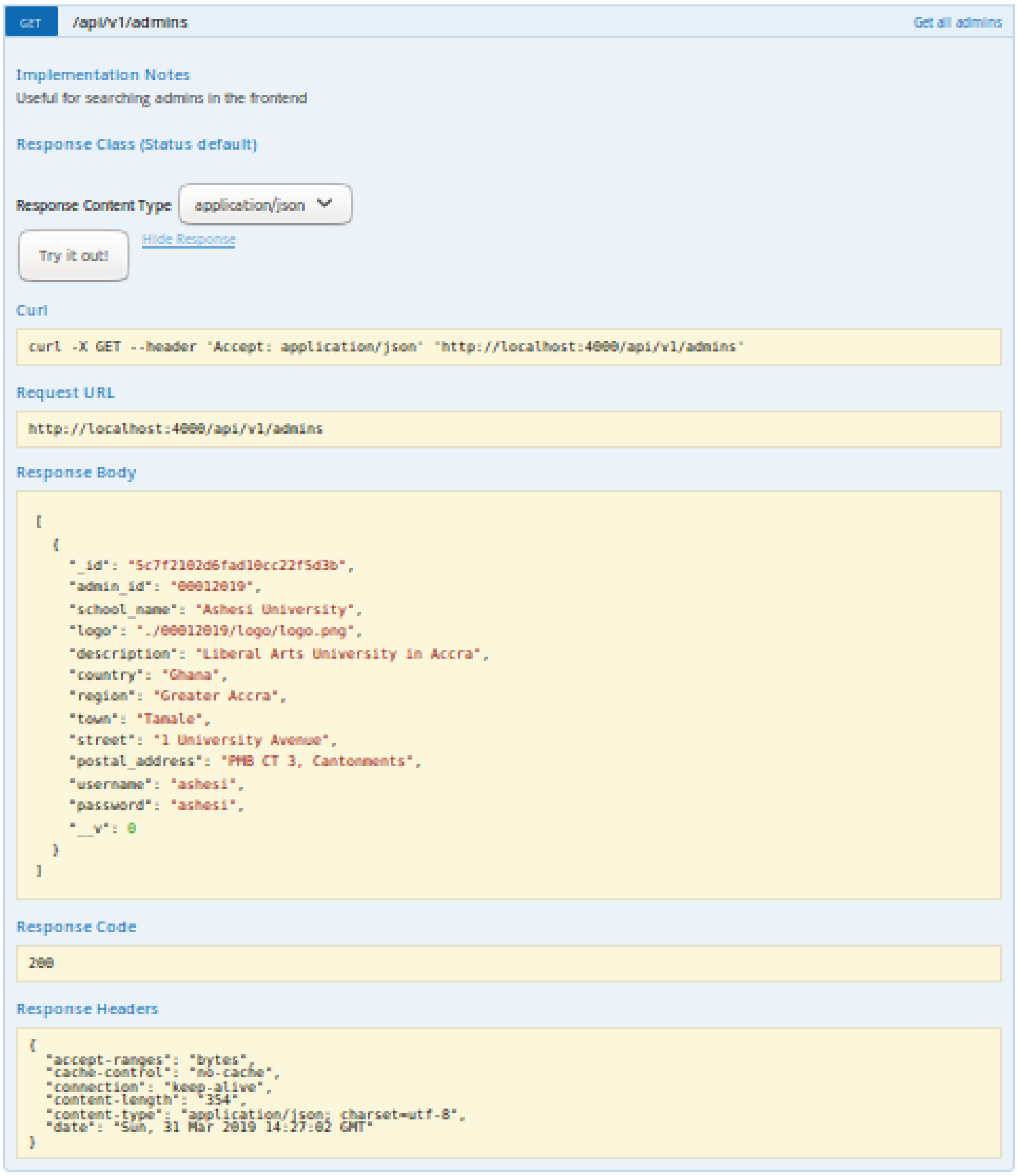


GET	/api/v1/students	Get all students
POST	/api/v1/students	Create a new student in the database
DELETE	/api/v1/students/{id}	Delete student with this id
GET	/api/v1/students/{id}	Get the student with this id
PATCH	/api/v1/students/{id}	Update/Edit student with this id

documentation#!/api/getApiV1FacultyId

Figure 17: 4.5.1 API documentation

Examples of API calls:



```
curl -X GET --header 'Accept: application/json' 'http://localhost:4000/api/v1/admins'
```

Request URL

```
http://localhost:4000/api/v1/admins
```

Response Body

```
[
  {
    "_id": "5c7f2102d6fad10cc22f5d3b",
    "admin_id": "00012019",
    "school_name": "Ashesi University",
    "logo": "./00012019/logo/logo.png",
    "description": "Liberal Arts University in Accra",
    "country": "Ghana",
    "region": "Greater Accra",
    "town": "Tamale",
    "street": "1 University Avenue",
    "postal_address": "PMB CT 3, Cantonments",
    "username": "ashesi",
    "password": "ashesi",
    "__v": 0
  }
]
```

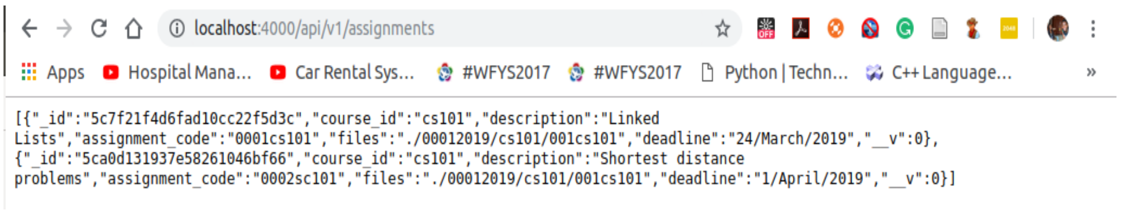
Response Code

```
200
```

Response Headers

```
{
  "accept-ranges": "bytes",
  "cache-control": "no-cache",
  "connection": "keep-alive",
  "content-length": "354",
  "content-type": "application/json; charset=utf-8",
  "date": "Sun, 31 Mar 2019 14:27:02 GMT"
}
```

Figure 18: 4.5.1 sample API call



```
[{"_id":"5c7f21f4d6fad10cc22f5d3c","course_id":"cs101","description":"Linked
Lists","assignment_code":"0001cs101","files":"./00012019/cs101/001cs101","deadline":"24/March/2019","__v":0},
{"_id":"5ca0d131937e58261046bf66","course_id":"cs101","description":"Shortest distance
problems","assignment_code":"0002sc101","files":"./00012019/cs101/001cs101","deadline":"1/April/2019","__v":0}]
```

Figure 19: 4.5.1 sample API call

4.5.2 Auto-grading Service

This component creates a secure environment to run unsafe student code. The environment is used to run facilitator's test on students' source code as well. The results of the student code or the results of the facilitator's test, when applied to students' code, is used to grade the student. To create a secure environment, a docker container is started from an ubuntu image with pre-installed compilers and interpreters. The Ubuntu operating system container creates a sandbox for running each student's code securely.

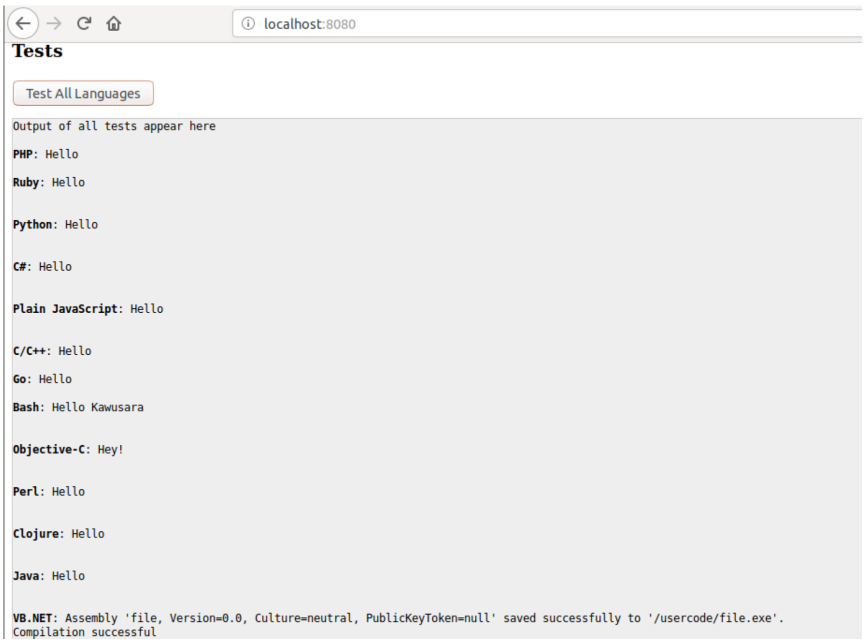
Below is the Docker file that creates the ubuntu image.

```
FROM chug/ubuntu14.04x64

# Update the repository sources list
RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main
universe" > /etc/apt/sources.list
RUN apt-get update

#Install all the languages/compilers we are supporting.
RUN apt-get install -y gcc
RUN apt-get install -y g++
RUN apt-get install -y php5-cli
RUN apt-get install -y ruby
RUN apt-get install -y python
RUN apt-get install -y mono-xsp2 mono-xsp2-base

RUN apt-get install -y mono-vbnc
RUN apt-get install -y npm
RUN apt-get install -y golang-go
RUN apt-get install -y nodejs

RUN apt-get install -y clojure1.4


#prepare for Java download
RUN apt-get install -y python-software-properties
RUN apt-get install -y software-properties-common

#grab oracle java (auto accept licence)
RUN add-apt-repository -y ppa:webupd8team/java
RUN apt-get update
RUN echo oracle-java8-installer shared/accepted-oracle-license-
v1-1 select true | /usr/bin/debconf-set-selections
RUN apt-get install -y oracle-java8-installer


RUN apt-get install -y gobjc
RUN apt-get install -y gnustep-devel &&  sed -i 's/#define
BASE_NATIVE_OBJC_EXCEPTIONS     1/#define
BASE_NATIVE_OBJC_EXCEPTIONS     0/g'
/usr/include/GNUstep/GNUstepBase/GSConfig.h


RUN apt-get install -y scala
RUN apt-get install -y mysql-server
RUN apt-get install -y perl

RUN apt-get install -y curl
RUN mkdir -p /opt/rust && \
    curl https://sh.rustup.rs -sSf | HOME=/opt/rust sh -s -- --
no-modify-path -y && \
    chmod -R 777 /opt/rust

RUN apt-get install -y sudo
RUN apt-get install -y bc

RUN echo "mysql ALL = NOPASSWD: /usr/sbin/service mysql start" |
cat >> /etc/sudoers
```

Shell Script to create the docker sandbox.

```
#!/bin/sh

###########################
# Docker SETUP            #
###########################
sudo apt-get update
sudo apt-get install -y docker.io
echo "Docker Setup complete"

###########################
# NodeJS setup            #
###########################
sudo apt-get update
sudo apt-get install -y nodejs
sudo apt-get install -y npm
echo "NodeJS setup Complete"

###########################
# Start Docker            #
###########################
sudo chmod 777 ../API/DockerTimeout.sh
sudo chmod 777 ../API/Payload/script.sh
sudo chmod 777 ../API/Payload/javaRunner.sh
sudo chmod 777 update_docker.sh

sudo systemctl unmask docker.service
sudo systemctl unmask docker.socket
sudo systemctl start docker.service
./update_docker.sh
```

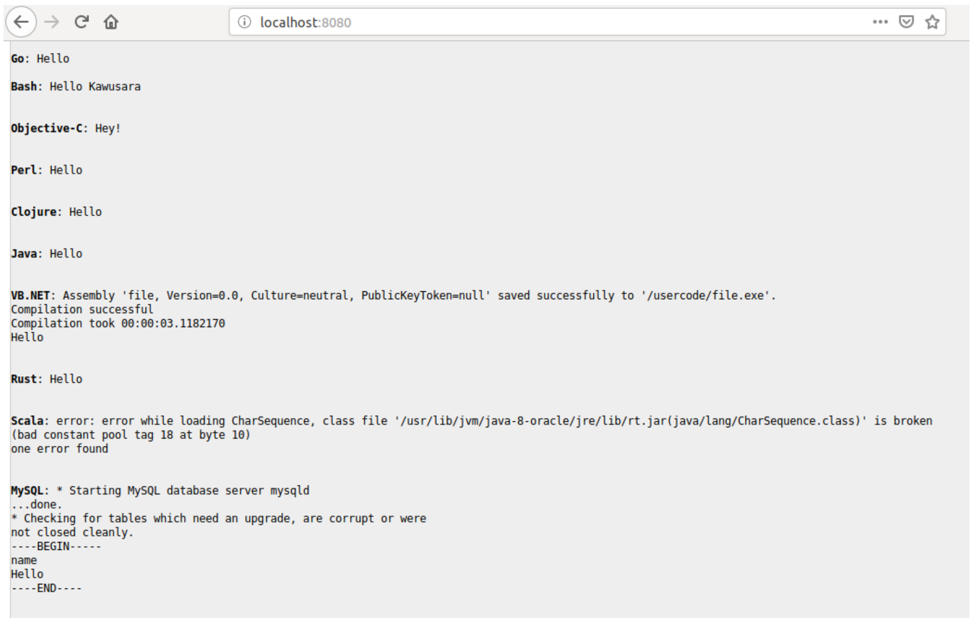
Testing the sandbox with hello program.



Tests

```
Output of all tests appear here

PHP: Hello

Ruby: Hello


Python: Hello


C#: Hello


Plain JavaScript: Hello


C/C++: Hello

Go: Hello

Bash: Hello Kawusara


Objective-C: Hey!


Perl: Hello


Clojure: Hello


Java: Hello


VB.NET: Assembly 'file, Version=0.0, Culture=neutral, PublicKeyToken=null' saved successfully to '/usercode/file.exe'.
Compilation successful
```

Figure 20: 4.5.2 Testing language support



```
Go: Hello

Bash: Hello Kawusara


Objective-C: Hey!


Perl: Hello


Clojure: Hello


Java: Hello


VB.NET: Assembly 'file, Version=0.0, Culture=neutral, PublicKeyToken=null' saved successfully to '/usercode/file.exe'.
Compilation successful
Compilation took 00:00:03.1182170
Hello


Rust: Hello


Scala: error: error while loading CharSequence, class file '/usr/lib/jvm/java-8-oracle/jre/lib/rt.jar(java/lang/CharSequence.class)' is broken
(bad constant pool tag 18 at byte 10)
one error found


MySQL: * Starting MySQL database server mysqld
...done.
* Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.
----BEGIN-----
name
Hello
----END----
```

Figure 21: 2.5.2 Testing language support

4.5.3 Plagiarism Component

This component is in caters for finding the peer plagiarism index between students who have submitted the same assignments.

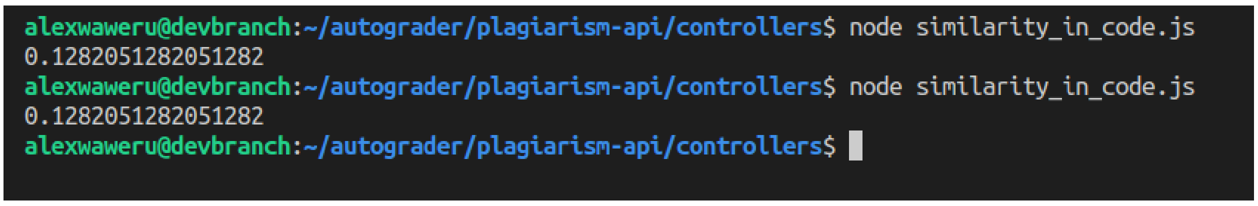
Sample source codes and their plagiarism index:

file1.txt

```
def bubbleSort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i])
```

file2.txt

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```



```
alexwaweru@devbranch:~/autograder/plagiarism-api/controllers$ node similarity_in_code.js
0.1282051282051282
alexwaweru@devbranch:~/autograder/plagiarism-api/controllers$ node similarity_in_code.js
0.1282051282051282
alexwaweru@devbranch:~/autograder/plagiarism-api/controllers$
```

Figure 22: 4.5.3 Plagiarism index from the 2 files

4.6 Implementation Techniques

During the implementation, the software system was broken down into four services, namely, the database API, the autograder service, the plagiarism-checker service, and the frontend client.

The database API is decomposed into model, controllers, and routes. The model contains the schema that models the application data. The controllers on the other side control the query logic and validation. The routes create the routes that would be used to expose the controller logic through http methods such as GET, POST, PATCH, and DELETE.

The autograder service is decomposed into two independent sub-systems. The first sub-system is the docker setup system that prepares the ubuntu image and pre-installs all the compilers and interpreters before running a docker container. The second sub-system is the autograding API that runs students code and facilitators' tests on the docker container.

The plagiarism checker contains a single route that takes two files as inputs and returns the plagiarism index between the two files. This component contains functions that calculate the structural and lexical similarity between two source codes.

These services allow for modular and independent development of each service. The frontend client brings the three other services together through API calls and http requests.

Chapter 5: Tests and Results

5.1 Overview

This chapter presents the various tests applied to the units and components of the system to establish that it satisfies the requirements of the intended system. The testing is divided into three categories; unit testing, component testing and system testing. The tests results are compared against the expected results and behaviour to establish if they pass the requirements.

5.2 Unit Testing

In unit testing, object classes are tested to establish that they produce the expected results or display the intended behaviour. In order to establish satisfactorily that the auto grading component supports the intended languages and it compiles them correctly we needed to write tests to be applied on the units in charge of language support and compilation. Below is a figure showing the results of the language support tests, database integrity tests and file operation tests.

```
Python file was saved!
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/38aa187bd0de0aaf8438":/usercode virtual_machine
/usercode/script.sh python file.py
------------------------------
DONE
ATTEMPTING TO REMOVE: temp/38aa187bd0de0aaf8438
------------------------------
Error file:

Main File
Hello!
*-COMPILEBOX::ENDOFOUTPUT-* .06
```

```
Time:
 .06

Data: received: Hello!

[sudo] password for alexwaweru: Clojure file was saved!
C/C++ file was saved!
C# file was saved!
Java file was saved!
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/e1dcf15a2d241f76e315":/usercode virtual_machine
/usercode/script.sh clojure file.clj
------------------------------
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/08093a26b6dc36324c50":/usercode virtual_machine
/usercode/script.sh 'g++ -o /usercode/a.out'  file.cpp
/usercode/a.out
------------------------------
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/206993939f5629590960":/usercode virtual_machine
/usercode/script.sh gmcs file.cs 'mono /usercode/file.exe'
------------------------------
Go file was saved!
Nodejs file was saved!
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/01492a9c5a58706f742e":/usercode virtual_machine
/usercode/script.sh javac file.java './usercode/javaRunner.sh'
------------------------------
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/5a57c14953baba551e6b":/usercode virtual_machine
/usercode/script.sh 'go run' file.go
------------------------------
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/17b72acc60dbf5f86b9b":/usercode virtual_machine
/usercode/script.sh nodejs file.js
------------------------------
DONE
ATTEMPTING TO REMOVE: temp/08093a26b6dc36324c50
```

```
------------------------------
Error file:

Main File
Hello*-COMPILEBOX::ENDOFOUTPUT-* .61

Time:
 .61

Data: received: Hello
DONE
ATTEMPTING TO REMOVE: temp/17b72acc60dbf5f86b9b
------------------------------
Error file:

Main File
Hello
*-COMPILEBOX::ENDOFOUTPUT-* .17

Time:
 .17

Data: received: Hello

DONE
ATTEMPTING TO REMOVE: temp/206993939f5629590960
------------------------------
Error file:

Main File
Hello
*-COMPILEBOX::ENDOFOUTPUT-* 1.23

Time:
 1.23

Data: received: Hello

DONE
ATTEMPTING TO REMOVE: temp/5a57c14953baba551e6b
------------------------------
Error file:

Main File
Hello*-COMPILEBOX::ENDOFOUTPUT-* 1.19

Time:
 1.19

Data: received: Hello
DONE
ATTEMPTING TO REMOVE: temp/e1dcf15a2d241f76e315
------------------------------
Error file:

Main File
```

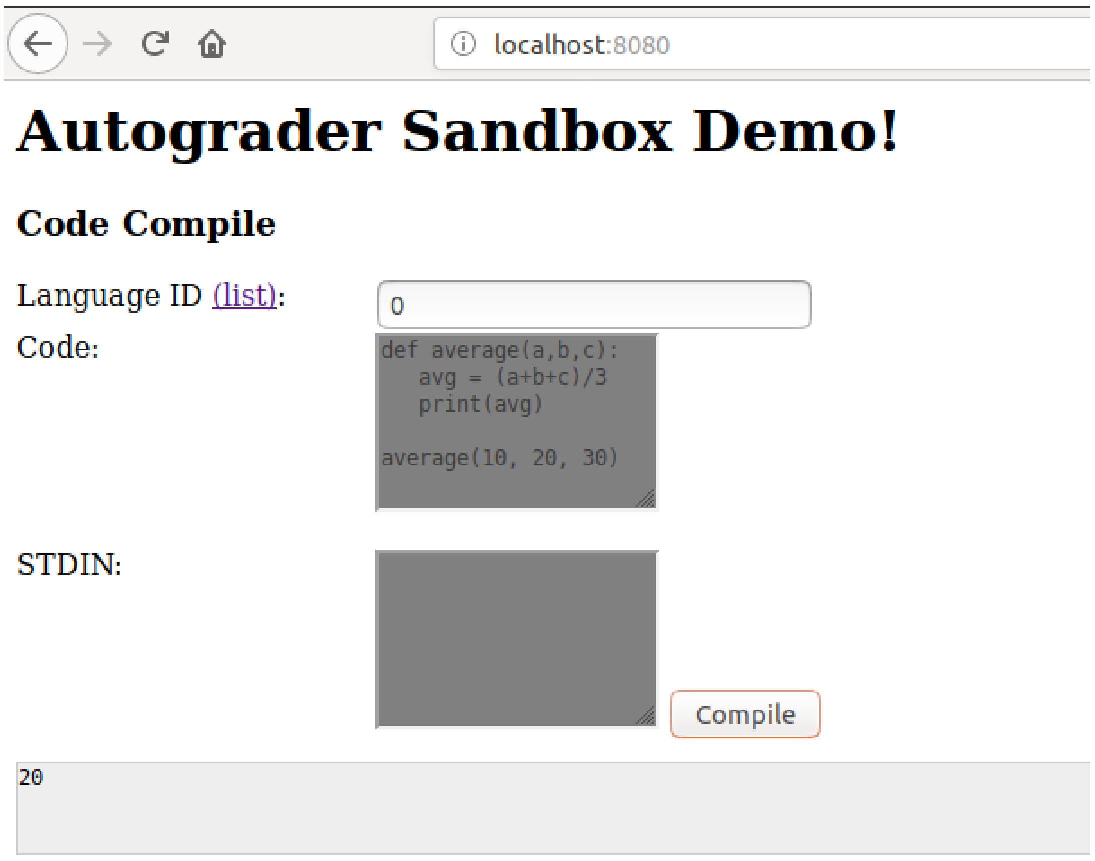
```
Hello
*-COMPILEBOX::ENDOFOUTPUT-* 3.61

Time:
 3.61

Data: received: Hello

DONE
ATTEMPTING TO REMOVE: temp/01492a9c5a58706f742e
------------------------------
Error file:

Main File
Hello
*-COMPILEBOX::ENDOFOUTPUT-* 3.28

Time:
 3.28

Data: received: Hello
```

5.3 Component Testing

A component is a logical module made up of various unit objects integrated together to perform a task. In component testing, the individual units are not tested but rather the emergent behaviour of the entire component. Component testing reveals integration errors between the various units. Below are a few test cases.

Test case: Running a python program from the front-end client.

- Precondition: A python program that calculates the average between three numbers. In this case the numbers are 10, 20 and 30.
- Expected results: The program should compile and print 20.
- Results: The front-end client should have 20 printed on the output box and backend should show a log of the details in compiling the program.



Autograder Sandbox Demo!

Code Compile

Language ID (list): 0

Code:

```
def average(a,b,c):
   avg = (a+b+c)/3
   print(avg)

average(10, 20, 30)
```

STDIN:

```
20
```

Figure 23: 5.3 Testing the autograding component

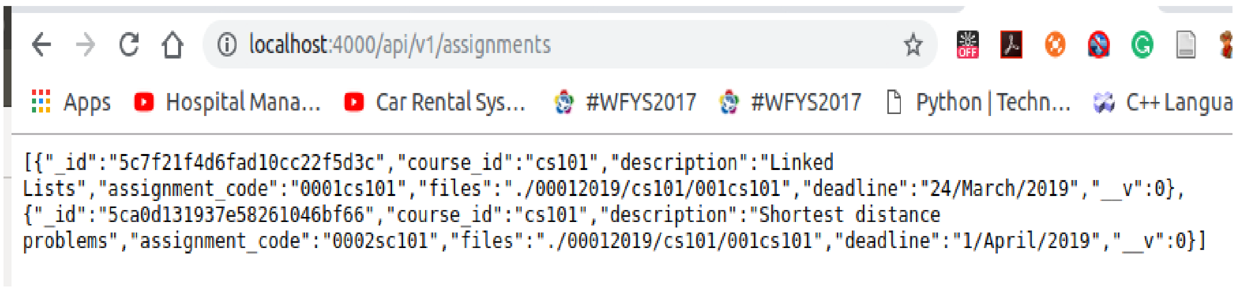
Below is the backend log of the front-end operation:

```
Input file was saved!
/home/alexwaweru/autograder/docker-api/API/DockerTimeout.sh 20s -
u mysql -e 'NODE_PATH=/usr/local/lib/node_modules' -i -t -
v  "/home/alexwaweru/autograder/docker-
api/API/temp/d02a22f276e64baf4788":/usercode virtual_machine
/usercode/script.sh python file.py
------------------------------
DONE
ATTEMPTING TO REMOVE: temp/d02a22f276e64baf4788
------------------------------
Error file:

Main File
20
*-COMPILEBOX::ENDOFOUTPUT-* .02

Time:
 .02

Data: received: 20
```

Test case: Making API call to view all assignments

- Precondition: Make an API call from the browser to display all assignments.

- Expected results: A list of assignments available.
- Results: The figure shows the results from the API call



```
[{"_id":"5c7f21f4d6fad10cc22f5d3c","course_id":"cs101","description":"Linked
Lists","assignment_code":"0001cs101","files":"./00012019/cs101/001cs101","deadline":"24/March/2019","__v":0},
{"_id":"5ca0d131937e58261046bf66","course_id":"cs101","description":"Shortest distance
problems","assignment_code":"0002sc101","files":"./00012019/cs101/001cs101","deadline":"1/April/2019","__v":0}]
```

Figure 24: 5.3 Testing the API component

5.4 System Testing

The API is the integration interface of the entire system. It contains endpoints that expose all the components and their functionalities. It exposes the database functionalities such as insertion and deletion as well as auto grading functionalities and the plagiarism checker. To test the system a front-end client was developed to consume a few of the API endpoints.

Chapter 6: Recommendations, Future Work and Conclusion

6.1 Recommendations

Below are a few suggestions that could improve the various components and the overall system.

- **Use Machine learning to train a model for detecting source code plagiarism**. The implemented plagiarism checker uses lexical and structural similarities between a pair of source codes to detect plagiarism. This method is ineffective for a large class size between it applies combinations to form the pairs. A better method would be to use pre-trained model to detect plagiarism. The model would require a lot of training and testing data which is hard to come by for this problem.
- **A desktop client** that tests the source on the students' machines. Running each students source code takes a lot of server time and it might stall other process. A solution to this is to create a distributed desktop application that runs the source code and tests on the students' machine and send the results to the server.

6.2 Future Work

Below are few requirements that we not completed within the time frame of the project and would therefore be implemented in a future version:

- **The front-end client.** Currently the front-end client consumes only about five endpoints of the API. In the future a more robust front-end client would be implemented to consume all the endpoints of the integration API.
- **Migrate the system to a cloud service.** Students generally submit assignments around the scheduled deadline. At that time the system will be resource intensive. To cater for the peak hour a lot of resources that are otherwise not in use most of the time will be needed. It is therefore much more economical to rent resources in

the cloud whenever you need them and release them whenever you do not. Migrating to the cloud will also allow the system to scale without having to purchase physical resources to cater for the scaling.

- **Test the efficiency of the Plagiarism checker.** Due to lack of data we could test the plagiarism checker to establish its accuracy. Currently, we rely on the algorithmic nature of the checker to check for correctness of the checker.

6.3 Conclusion

This project is an attempt to create a system that grades programming assignments as well as detects any plagiarism in the source code. To build the system, we divided into logical components, namely; autograding component, plagiarism component and the database. We created an API to integrate all these components together. The autograding component is built on top of a Docker container to provide a secure sandbox. The plagiarism component implements an algorithm that quantifies the lexical and structural similarities between pair of source code. The API is build using Node.js and documented using the Swagger npm module. Finally, to demonstrate the functionalities of the system a front-end client that consumes a sample of the API endpoints is built. This project is a proof that we can automate the grading of programming assignments as well as check for source code plagiarism.

BIBLIOGRAPHY

[1] Peter E. Coughlin. 2015. Plagiarism in five universities in Mozambique: Magnitude, detection techniques, and control measures. *International Journal for Educational Integrity* 11, 1 (June 2015), 2.

[2] Zoran Đurić and Dragan Gašević. 2012. A Source Code Similarity System for Plagiarism Detection. *The Computer Journal* 56, 1 (March 2012), 70–86. DOI:https://doi.org/10.1093/comjnl/bxs018

[3] David Kay, Terry Scott, Peter Isaacson, and Kenneth Reek. Automated Grading Assistance For Student Programs. Retrieved from https://www.researchgate.net/profile/Kenneth_Reek/publication/221537823_Automated_grading_assistance_for_student_programs/links/0c96052e35a97e7321000000/Automated-grading-assistance-for-student-programs.pdf

[4] Alan Parker and James O. Hamblen. 1989. Computer Algorithms for Plagiarism Detection. IEEE Transactions on Education, 32(1), (Jan. 1989).

[5] David J. Malan. 2013. CS50 sandbox: secure execution of untrusted code. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 141-146. DOI: https://doi.org/10.1145/2445196.2445242

[6] Fatima Abu Deeb and Timothy Hickey. 2015. The Spinoza code tutor: faculty poster abstract. J. Comput. Sci. Coll. 30, 6 (June 2015), 154-155.

[7] Georgina Cosma and Mike Joy. 2012. An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis. IEEE Transactions on Education, 61(3), (Mar. 2012).

[8] Stephen Burrows, S. M. M. Tahaghoghi and Justin Zobel. 2007. Efficient Plagiarism Detection for large code repositories. Wiley InterScience 37, (Sept. 2007), 151-175. DOI: 10.1002/spe.750

[9] Susilo V. Yulianto and Inggriani Liem. 2014. Automatic Grader for Programming Assignment Using Source Code Analyzer. IEEE, 14, (Mar. 2014), . **DOI:** 10.1109/ICODSE.2014.7062687