Software Testing Plan

Student Code Online Review and Evaluation (2.0)

S.C.O.R.E. (2.0) aims to enhance and extend the original S.C.O.R.E. application. Adding functionality in server connections, grading with rubrics, and cheat detections are all intentions to deploy the system into CSE classrooms.

Team Members

Dorothy Ammons - dammons2022@my.fit.edu
Patrick Kelly - pkelly2022@my.fit.edu
Shamik Bera - sbera2022@my.fit.edu
Rak Alsharif - ralsharif2021@my.fit.edu

Faculty Advisor and Client

Raghuveer Mohan

November 23, 2025

Table of Contents

1. Introduction

The main purpose of this document is to test the detections, such as the MOSS and AI, ensuring that they successfully detect collusion and AI generation on the students' submissions. We will also test the terminal connections to ensure that students and professors can access the SCORE platform through command line inputs.

2. Test Plan

The functional requirements from the SRS document will be tested using outlined test cases before implementing them in the application. Those requirements include the MOSS detection, AI detection, custom rubrics, importing rosters, exporting grades, and the Remote Command Line Server Connection via HTTPS connection. That connection through the terminal is also where we use test cases through the code written in Python.

3. Functional Test

3.a Import Rosters

- Import CSV file: Professor can import a Canvas roster CSV file into a selected class over HTTPS
    - Test 1: On the web app, as a professor user, create a class, click the *import* button, and upload a valid Canvas-exported CSV file. The roster should be occupied with the students from the file.
    - Test 2: On the HTTPS connection, as a professor user, send a POST /class/{id}/roster request with a valid Canvas CSV file. The roster should be occupied with the students from the file.
    - Test 3: On the web app, as a professor user, upload a CSV file containing a duplicate student already in the class roaster. The student should not be duplicated in the roster.
    - Test 4: On the web app, as a professor user, attempt to upload an invalid file type. The system should reject the upload with an error message displayed.
    - Test 5: On the HTTPS connection, as a professor user, send a corrupted CSV file. The system should reject the file with an error message displayed.
    - Test 6: On the web app, as a professor user, upload an empty CSV file. The system should display an error message and not add any students.
    - Test 7: On the HTTPS connection, as a professor user, attempt to send a CSV import request without selecting a class. The system should reject the request with an error message displayed.

3.b Export Grades

- Export Grades CSV file: Professor can export student grades for a given assignment in Canvas on a CSV format
    - Test 1: On the web app, as a professor user, navigate to the assignment page and click the *Export Grades* button. A CSV file should be created containing all student grades for that assignment in the correct format for Canvas.
    - Test 2: On the HTTPS connection, as a professor user, send a GET /assignment/{id}/grades/export request for a valid assignment. A CSV file should return all the students' grades on Canvas with the correct format.
    - Test 3: On the web app, as a professor user, attempt to export grades for an assignment that has no submissions. The system should still create a CSV file with student names, but shows empty grade fields.

3.c MOSS Similarity Detection

- MOSS API: Professors can use it to run similarity detection across submissions with configurable similarity thresholds
    - Test 1: On the web app, as a professor user, navigate to the assignment page and click the *Run Similarity Detection* button. The system should call the MOSS API and return similarity scores for all submissions.
    - Test 2: On the HTTPS connection, as a professor user, send a POST /assignment/{id}/similarity request with a valid threshold parameter. The system should return similarity scores for all submissions in the assignment.
    - Test 3: On the web app, as a professor user, set a similarity threshold. The system should flag all submissions with similarity above the threshold as potential collusion.
    - Test 4: On the HTTPS connection, as a professor user, send a request with an invalid threshold input. The system should reject the request with an error message displayed.
    - Test 5: On the web app, as a professor user, run similarity detection on an assignment with no submissions. The system should return a message showing that there are no results displayed.
- MOSS Score visualization: Professor can view similarity scores in a matrix with threshold-based color highlights
    - Test 1: On the web app, as a professor user, run similarity detection on an assignment with submissions. The system should display an n by n matrix where rows and columns represent student names and cross sections show similarity percentage.
    - Test 2: On the web app, as a professor user, configure a similarity threshold. The visualization should highlight scores above 80% in bright

red, scores below 20% in green, and values near 80% in shades of yellow/orange.
    - Test 3: On the web app, as a professor user, run similarity detection on an assignment with only one student submission. The system should display a matrix with one row and column, containing no similarity values.
    - Test 4: On the web app, as a professor user, run similarity detection with multiple submissions where all scores are below the threshold. The matrix should render all similarity values in shades of green.
    - Test 5: On the web app, as a professor user, run similarity detection with multiple submissions where all scores exceed the threshold. The matrix should highlight those scores in shades of red.
    - Test 6: On the HTTPS connection, as a professor user, send a GET /assignment/{id}/similarity-visualization request. The system should return JSON data representing the n by n matrix with similarity scores and threshold-based highlight values.

3.d AI Detection

- AI detection on submissions: Professors can use LLM model to detect the possibility of AI-generated submissions
    - Test 1: On the web app, as a professor user, navigate to the assignment page and click the *AI Detection* button. The system should run the LLM and return a table with each student's probability percentage.
    - Test 2: On the HTTPS connection, as a professor user, send a POST /assignment/{id}/ai-detection request. The system should return a JSON response plotting each student to their probability percentage.
    - Test 3: On the web app, as a professor user, run AI detection on an assignment with multiple submissions. The system should display a table with all student names with each of their percentages.
    - Test 4: On the HTTPS connection, as a professor user, send a request for an assignment with no submissions. The system should display a message "No submission available" with an empty result set.

3.e Custom Rubrics

- Custom Rubric Creation: Professor can define grading rubrics with points and penalties to the submissions
    - Test 1: On the web app, as a professor user, navigate to an assignment and input total points of one or higher. The system should save the rubric with entered total points.

- Test 2: On the web app, as a professor user, assign points to individual test cases. The rubric should save successfully and show the assigned points for each test case.
- Test 3: On the web app, as a professor user, allocate more points to the test case than the overall assignment total, such as extra credit. The system should accept the rubric and save it correctly.
- Test 4: On the web app, as a professor user, allocate points for attempts, compilation, and time limits. The rubric should save and apply these factors in grading.
- Test 5: On the web app, as a professor user, configure late submission penalties. The system should apply the penalty when submissions are graded.
- Test 6: On the HTTPS connection, as a professor user, send a POST /assignment/{id}/rubric request with valid rubric data. The system should save the rubric and confirm success.
- Test 7: On the HTTPS connection, as a professor user, send a POST /assignment/{id}/rubric request with invalid rubric data. The system should reject the request with an error message.

4. User Test

4.a HTTPS Connection

4.a.1 Professor

- Logs into the system through the terminal, which opens a browser and authenticates the user through OAuth. The professor should be able to view their list of classes, create a new class, and remove an existing class. When the class is selected, the professor should upload a CSV roster to add students. The professor should then view the list of assignments, add a new assignment, remove an assignment, and export the grades for an assignment in CSV format to use in Canvas. The tester will be asked if they could accomplish each task, and rank the ease of completing each task.

4.a.2 Student

- Logs into the system through the terminal, which opens a browser and authenticates the user through OAuth. The student should view their list of classes, select a class, and view the assignments for that class. Upon selecting an assignment, the student should review the displayed information about the requirements and due

dates, then submit a code solution as a file. After submission, the student should check the status of their code to view their current grade and feedback based on failed test cases. Then, the student should make another submission so the new attempt overwrites the previous submission. The tester will be asked if they could accomplish each task, and rank the ease of completing each task.

4.b Web App

4.a.1 Professor

- Log in to the system, navigate to a class, and ensure an option to import rosters is on the created class page. Within the selected assignment, the professor should confirm that rubric options are available along with test cases, then click the *Detect Similarities* button, enter a threshold, and view the visualization of MOSS similarity scores. Then, the professor should click the *Detect AI* button and verify that a probability score table is generated with each student's likelihood of AI-generated work. Finally, the professor should export the grades for the assignment in CSV format. The tester will be asked if they could accomplish each task, and rank the ease of completing each task.