Project Title: Student Code Online Review and Evaluation

Names and email addresses of team members:
Charlie Collins, ccollins2021@my.fit.edu
Thomas Gingerelli, tgingerelli2021@my.fit.edu
Logan Klaproth, lklaproth2021@my.fit.edu
Michael Komar, mkomar2021@my.fit.edu
Faculty advisor from CSE: Raghuveer Mohan, rmohan@fit.edu
Client name and affiliation: Raghuveer Mohan, CSE Professor
Date(s) of Meeting(s) with the Client for developing this Plan:
1/22/2025

Goals:

- Develop a more robust submission platform
  - Automatically grades student submissions
  - Provides project-specific feedback based on test cases
  - Ability to accept and run multiple file/program types
- Streamline the submission and grade input process
  - Student authentication
  - Canvas integration
  - Determines the similarity of submissions
- Develop skills in full-stack development
  - Interactive GUI on student and professor side
  - Back end for automatic grading
  - Storage of submissions and assignment files

Motivations:

From the student's perspective, the current authentication process is a pain point as we must register and save an access token, increasing the steps required to submit an assignment. From the student perspective, the current submit server does not allow for immediate feedback on our assignment performed in the autotests, which provides students with less knowledge about how they are performing.

From the professor's perspective, it can be a pain point to have to manually enter grades into canvas, once the submissions have been graded, so canvas integration eases this by automatically submitting the grades to canvas. Additionally, assignments submitted directly to canvas can be placed into folders or have their name changed by canvas. This can make running auto testing scripts difficult. However, other sites, like Kattis, that can have auto testing, don't allow for other necessary features like uploading assignments.

Approach:

Auto Testing

Student

- The student will have their assignment submissions tested against the hidden test cases created by the professor

Professor

- The professor can configure the auto-testing environment to run student submissions
- The professor can supply both sample, and hidden test cases to be run by the auto test
- The professor will be able to view how many test cases a student's submission has passed
- The professor can set bounds on the amount of time a solution can take to run on a particular problem

Immediate Feedback

Student

- The student can view the exact number of test cases that were passed and how many were failed
- The student can view professor feedback associated with a particular test case that they failed
- The student can view previous submission attempts auto test scores to determine if their work has improved

Professor

- The professor can create helpful feedback to attach to a test case in order to help students understand where they went wrong
- The professor can select how much feedback their students can receive from an assignment submission

Grading Portal Integration

Professor

- The professor can manually adjust the grade of an auto tested assignment
- The professor can set a curve to be automatically applied to the grades
- The professor can have the assignment grades automatically uploaded to canvas after the due date has passed
- The professor can export the assignment grades as a .CSV file

MOSS integration

Professor

- The professor can view a student submission's MOSS similarity score to other submissions
- The professor can view what submissions are flagged as similar
- The professor can set a threshold similarity score to flag a submission
- The professor can provide starter code to be ignored by MOSS

User Authentication

- The user can access the platform using their provided Florida Tech email address
- The user will not have to register and maintain a control code for assignment submission

Shell Client

- The user can interact with the application through a command line interface in which they can do the following:

  Student

  - The student can upload C, C++, Java, and Python files to a class assignment
  - The student can view the grade of a submission after the due date passes
  - The student can view the breakdown of hidden test cases that they passed or failed for each submission
  - The students can view written professor feedback associated with each hidden test case that they failed

  Professor

  - The professor can create new assignments and test cases, as well as associated feedback, for students
  - The professor can receive and view student submissions as well as their scores from the auto test
  - The professor can view the MOSS plagiarism score for each student submission

Web-App

- The user can interact with the application through a graphical interface on the web in which they can do the following:

  Student

  - The student can navigate through their classes and assignment descriptions
  - The student can upload C, C++, Java, and Python files to a class assignment

- The student can view the breakdown of hidden test cases that they passed or failed for each submission as well as associated professor feedback

Professor

- The professor can create new assignments and test cases, as well as associated feedback, for students.
- The professor can receive and view student submissions as well as their scores from the auto test.
- The professor can view the MOSS plagiarism score for each student submission

Novel features/functionalities:

Assignment Creation

While other platforms, such as Kattis, allow for automated grading with instant user feedback, they don't provide the user (Professor) to upload new assignments for their students to work on.

Test Case feedback

From our survey of other relevant applications, we did not find any that allow the professor to attach feedback to specific test cases. We feel this feedback has the potential to increase a student's ability to learn from their errors.

Automatic MOSS calculation

The moss algorithm is a winnowing hashing algorithm that calculates a similarity score between text files. While Stanford offers an implementation of this algorithm, to our knowledge, there is no application that automatically calculates the MOSS score between all submitted files of an assignment, making it easier to catch cheating.

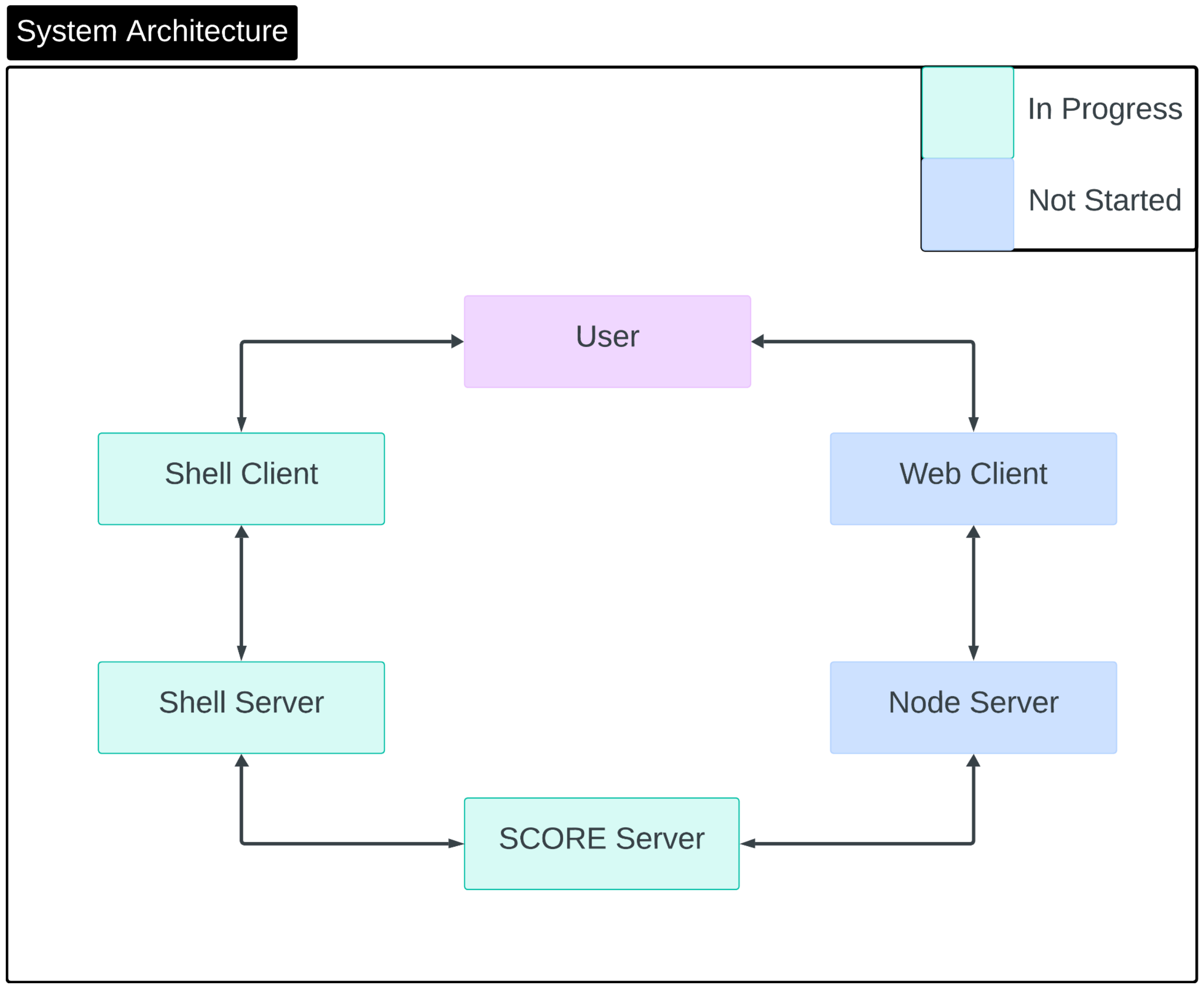
Algorithms and Tools: Potentially Useful Algorithms and Software Tools

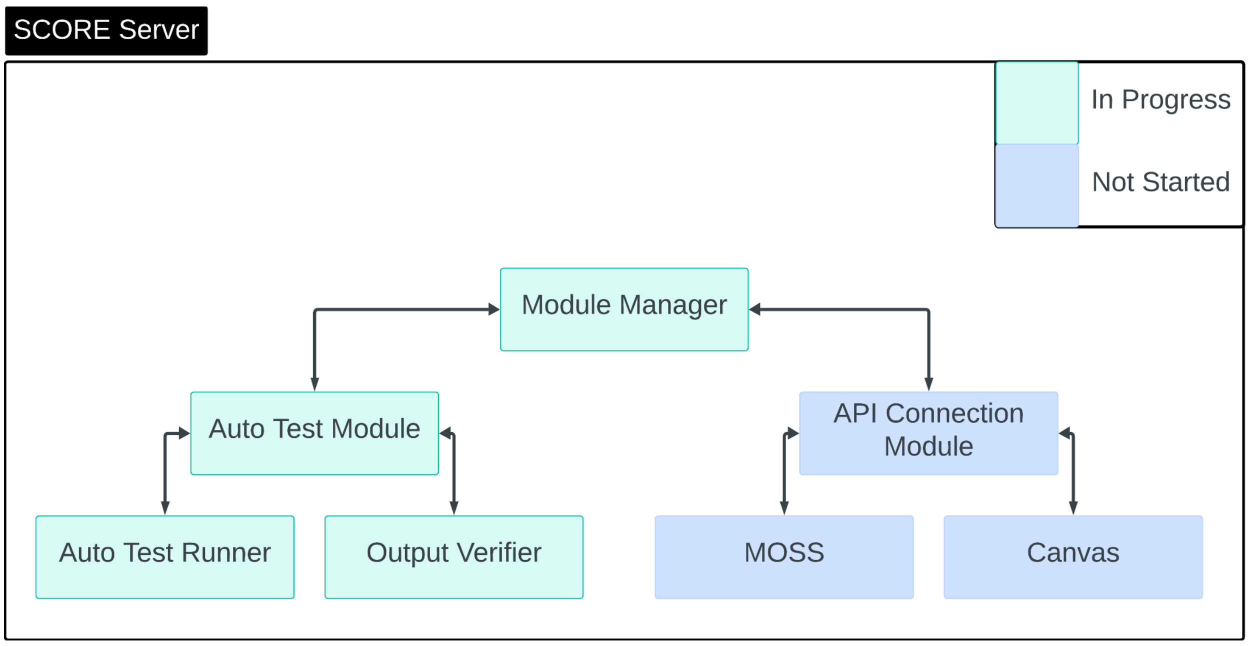
- MERN: Full Stack
  - MongoDB
  - Express
  - React
  - Node JS
- Canvas API
- MOSS
- AWS Cloud Services
- Rust
- Python
- Docker
- SFTP
- OAuth

Technical Challenges:

1. Communication between server component without concurrent modifications
2. Atomicity of server actions
3. Student user authentication using OAuth

Design:





Evaluation:

Load testing:

Since the intention of this project is to create an application that will be used within a classroom setting, it is vital that it is able to handle the load of a class. To test for this, we plan to simulate having concurrent users across the shell client and the web application. We will measure this through the max number of concurrent users without the server crashing.

Test accuracy:

A primary functionality of SCORE is to evaluate student's code, thus it is very important we ensure that code can be properly run and evaluated. To do this, we plan on creating sample assignments and test cases, then submitting code to those assignments. We will then ensure that the code passes the correct number of test cases. We will measure accuracy by keeping the number of correctly scored submissions out of the total number of submissions, with the goal being to keep this percent as close to 100% as possible.

User demo:

In order to ensure that SCORE is user friendly, we plan to conduct a trial run of the application within Dr. Mohan's class. This will start with having Dr. Mohan create a class, add students to the class, and create an assignment with test cases. Then, we will open up SCORE to the class and have them access the new assignment, submit their solution to it, and view the feedback from the auto test. The hope of this trial run is to reveal any pain points in the use of the software that can be addressed before the final milestone. We plan to measure this through a survey where we have the users rank the features usability from 1 to 10.

<u>Progress Summary:</u>

Module/feature	Completion %	To do
Auto Testing	80%	Auto Test caller, output comparisons, locating files
Immediate Feedback	50%	Implement professor-provided feedback
Grading Portal	0%	Not started
API Integrations	0%	Not started
User Authentication	0%	Not started
Shell Client	90%	Implement a man page and more robust responses
Web app	0%	Not started
SCORE server	80%	Integrate components

Milestone 4 (Feb 24): itemized tasks:

- Finish implementing, testing, and demoing auto testing and feedback
- Implement, test, and demo the front end of the web application
- Implement, test, and demo user authentication
- Implement, test, and demo server component integration

Milestone 5 (Mar 26): itemized tasks:

- Implement, test, and demo the web app backend
- Implement, test, and demo grading portal
- Conduct evaluation and analyze results
- Create poster for Senior Design Showcase

Milestone 6 (Apr 21): itemized tasks:

- Implement, test, and demo Canvas integration
- Implement, test, and demo Moss integration
- Test/demo of the entire system
- Conduct evaluation and analyze results
- Create user/developer manual
- Create demo video

<u>Task Matrix for Milestone 4 (teams with more than one person)</u>

Task	Tommy	Logan	Michael	Charlie
Finish auto testing and feedback	50%	50%	0%	0%
Web App Front End	0%	50%	25%	25%
User authentication	40%	0%	30%	30%
Integrate Server components	0%	0%	50%	50%

<u>Description (at least a few sentences) of each planned task for Milestone 4:</u>

<u>Task1:</u> The first task of milestone 4 is to complete auto testing and feedback. Currently this feature is able to create a docker image, and spin up the container from this image. It can place submitted code into this container, and retrieve the output of that code. To complete this feature, we need to finish the server component that will handle auto testing. This will maintain a queue of submissions that need to be tested and it will execute the auto testing on that code. Additionally, we will finish the checker portion of auto testing to determine the correctness of the results.

<u>Task 2:</u> The second task of this milestone is to begin the work into the web app portion of this project by creating the front end. This will be created using the React framework, and will not include any backend functionality. We plan to create both a student and professor view with all front end features working.

<u>Task 3:</u> The third task of this milestone is to start working on user authentication. At this point in the project, we do not have any sort of sign in or register, which is the goal of this task. To accomplish this, we plan to use google OAuth, with the school provided Florida Tech emails. This will save students from having to maintain separate login information, while not having to deal with the complexities of the TRACKS login.

<u>Task 4:</u> The final task of this milestone is to integrate the server components. At this point, the team has finished most of the functionality of the SCORE server. However, these functionalities exist as separate components that have been run individually. In this task, we aim to integrate all of these components into one server, thus creating a fully functioning SCORE server.

Approval from Faculty Advisor

"I have discussed with the team and approved this project plan. I will evaluate the progress and assign a grade for each of the three milestones."

Signature: ________________________________ Date: ________
