SCORE Milestone 4 Project Evaluation

Team Members:
Charlie Collins, ccollins2021@my.fit.edu
Michael Komar, mkomar2021@my.fit.edu
Logan Klaproth, lklaproth2021@my.fit.edu
Tommy Gingerelli, tgingerelli2021@my.fit.edu

Faculty advisor/client:

- Dr. Mohan - rmohan@fit.edu

Milestone 4 Progress

Task	Completion	Charlie	Logan	Michael	Tommy	To Do
Finish Auto Test	90%	20%	40%	0%	40%	Handle custom verifiers.
Implement user auth	100%	50%	0%	50%	0%	N/A
Implement web app front end	75%	35%	35%	30%	0%	Sign in component, view submission component.
System Integration	90%	60%	0%	20%	20%	Handle file transfer.

Discussion of accomplished tasks:

Task 1 (Finish Auto Test): The auto test module was started during the last milestone, resulting in the functionality to create a docker container that runs a user specified program with user specified input. To finish the auto test module, this milestone we added the ability to save the output from the docker to a file, and created a python server to manage the auto test. Saving to a file external to the docker allows the server to gather the feedback on a submission. This includes not only what tests were passed or failed, but also the professor specific feedback that is associated with a failed test case. This feedback is then written to a file so that it can be viewed through either interface. The python server is used to maintain a queue of files that need to be tested, and will pass them to the docker module as resources become available. This server also waits for assignments to be finished running before passing the output to the feedback module which will create the feedback and write it to a file.

Task 2 (Implement User Auth): This task involved creating functionality to verify who was using the application. Up to this point, for development purposes, we just had the ability to give a user's name to "login" to the application as that user. However, with this task, we implemented Google OAuth to verify the users. OAuth is an open authorization protocol that we leveraged for authentication. Essentially, on startup, the application makes a request to the Google OAuth server and opens a web browser on the client's machine to the OAuth page. The user then follows Google's steps to sign in and authorize SCORE. After this, the application receives an id token that we can eventually exchange with Google's server to access some user information. Of this information, SCORE uses the user's email. Since this email is the primary key of our user database, once we receive the user's email from Google, we can sign the user in.

Task 3 (Implement Web App Front End): From the beginning, the intention of the project was to have two interfaces, a command line shell interface as well as a web app interface. Up to this point, we had only made development efforts to the command line interface, so with this task, we are starting the web app interface by implementing the front end. To implement the front end, we used the React javascript framework. React works through the use of components written in jsx, which allows us to easily divide the work of the front end. The primary focus of our development efforts was to implement all of the main functionality, which we were able to accomplish. On the professor end, we made the create assignment component as well as the professor dashboard component. On the student end, we made the view assignment, submit assignment, and student dashboard components. However, there are some stylistic and UI improvements that we still plan to make going forward.

Task 4 (System Integration): With the completion of the auto test and user authentication modules, there was a need to have dedicated development time to system integration. This was because these modules, while working, were working independently, so the goal of this task was to make these components communicate with each other, so that they act as one server. The main components that we integrated were the Rust shell server, the python auto test manager, and the user authentication functionality. The Rust server and user authentication were integrated through multithreading of the main Rust thread. We then used some semaphores and a mutex guarded string to communicate the id token retrieved by the OAuth with the main rust thread. The other main integration was between the Rust server and the python auto test manager. This was done with a tcp connection, as that was the most efficient way to communicate between these two separate modules. On the Rust side, it maintains a list of user submitted files, and sends a tcp message to the python manager every 30 seconds containing all the files in the list. On the python side, a separate thread was used to monitor this tcp stream and to add the files to the queue of files to be tested. The only work left to do is some integration with the sftp module that handles the file transfers.

Discussion of member contribution:

Charlie: This milestone I played a part in most of the tasks that were being worked on. The first task that I worked on was user authentication. I worked with Michael to create a working demo of OAuth in a cli application. Once we had this done, I worked on making this demo into a function that could easily be ported into the rust client. After user authentication, I began working on the front end. I made the create assignment, view assignment, submit assignment, and user components, as well as some odds and ends as needed. Next I worked on the auto test task. While this was mostly completed by Logan and Tommy, I created the thread that listens to the tcp stream and adds files to the queue. Finally, as the milestone came to end, I worked on the integration task. I created a new thread in the rust server that maintained the list of submitted files and sent them in a tcp stream to the python manager.

Michael: At the beginning of the milestone, Charlie and I combined our efforts to create a working demo of Google OAuth in a command line application. After we finished this, I worked on storing the client ID and client secret in an env file for security before it was integrated into the rust client. After user authentication, I worked on the front end for the web application. The components that I made were the card list, student card, and professor card components. These make up the main portion of the student and professor dashboard.

Tommy: I spent this milestone working on the python manager for the auto test portion of the application. This involved making separate threads for the auto test handler and the feedback handler. The auto test handler contains a loop that iterates through each submitted file and makes an auto test object, constructed with the appropriate information to run the auto test. This

object is then moved to a thread where its main function is executed. Similarly, the feedback handler iterates through a list of tested files, and creates a feedback object for each one. This object is then moved to its own thread where its main function is executed. This process will result in the feedback for the submission to be contained in the description json file for that submission.

Logan: I started this milestone working on the auto test task. My main contribution was to the auto test object. In the previous milestone, I created a python script that would create a docker container, and run a user specified file. For this milestone, I converted this script to an object to make it easier to be called by the python manager. I also added the functionality to save the output to a user specified output file, so that feedback can be generated. After I completed my portion of the auto test task, I went on to work on the web app front end. The components I completed were header, navigation, and dashboard. On top of that, I handled the routing for the application so that it switches between the different components properly.

Task Matrix for Milestone 5:

Task	Charlie	Logan	Michael	Tommy
Implement web app back end	40%	20%	40%	0%
Implement grading portal	20%	20%	0%	60%
Conduct evaluation and analysis	25%	25%	25%	25%
Create senior design poster	10%	40%	40%	10%

Discussion of Milestone 4 Tasks

Task 1 (Implement web app back end): With the front end of the web application being completed in the previous milestone, the next step is to implement the backend of the web app interface. This backend will be implemented in Node.js, and will interact with the SCORE server. We plan to accomplish this by having the node server act similar to the rust server. This means that it will call the appropriate python modules when the user interacts with them in the front end, as well as notifying the python manager on new submissions. With the completion of this task, both of the interfaces to the application will be complete.

Task 2 (Implement grading portal): In the previous milestone we completed the auto testing portion of the application. This means that all submissions will now be graded, so we need to make the functionality for the professor to interact with these grades. This means not only

viewing the score, but also being able to view the source code, modify the grades, and apply an assignment wide curve. This will also be the place where the professor will be able to upload the grades to canvas, as well as view the MOSS scores, although these will both be done at the next milestone.
Task 3(Conduct evaluation and analysis): At this point, the main functionality of the SCORE application will be complete. This means that we will be able to start conducting our evaluation and then analyze the results. The evaluations we will conduct will be testing the accuracy of our auto testing component by running code that we know whether should pass or fail. It is imperative that the accuracy be as close to 100% as possible. We also want to demo SCORE to users and get feedback. We have talked with Dr. Mohan about doing a trial run in his class, but depending on the progress at the time of this testing, we might limit the demo to a few students and work closely with them to get feedback.
Task 4 (Create senior design poster): As we near the senior design showcase, we will need to make a poster. We want this to contain images of our interfaces, as well as showing off the evaluations, so we will wait until the end of this milestone to make this poster.

Dates of meetings with the client/advisor:
2/5/2025 at 4 pm
2/19/2025 at 4 pm

Client/Advisor feedback
Task 1 (Finish Auto Test):

- It will be sufficient to just allow for a set number of submissions languages
- The professor should be able to specify which languages are to be accepted
    - This can be multiple languages
- Ensure that outputs can be tested with a professor provided verifier
    - Not everything will be scored using diff

Task 2 (Implement User Auth):

- My only concern is that some professor may not have google accounts
    - Consider allowing other types of email than school emails or other types of authentication

Task 3 (Implement web app front end):

- Create assignment:
    - Make the configure auto test a checkbox
    - Make the number of submissions infinite by default
    - Make the input for the number of submissions a text box rather than a drop down
    - Make it possible to upload a text file for the test cases
    - Add a verifier column

Task 4 (System Integration):

- None given

Faculty Advisor Signature: _____

Date: _____

Evaluation by Faculty Advisor

- **Faculty Advisor: detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu**
- **Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)**

Charlie Collins	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Tommy Gingerelli	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Michael Komar	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Logan Klaproth	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10