SCORE Milestone 2 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 2 Progress

Task	Completion	Charlie	Logan	Michael	Tommy	To Do
Implement the Shell Application	50%	20%	15%	50%	15%	Client-Server integration (next milestone)
Implement Assignment Creation	80%	15%	20%	15%	50%	Fix directory names and import statements on recent branch.
Implement Assignment Submission	80%	40%	20%	20%	20%	Use SFTP to transfer file from client to server
Implement Assignment View	100%	20%	50%	20%	10%	Generalize to “View” both assignments and classes using flags to differentiate.

Discussion of accomplished tasks:

Task 1 (Implement the Shell Application): The primary objective of this milestone was to begin creation of the shell which houses all of the modules to be accessed by the different types of users. The shell, written in rust, has access to the custom commands defined in the below tasks as well as access to native bash commands by passthrough. We define the process of creating a module as implementing functionality in python and then creating a matching command keyword in the shell to run the command. The decision to write the shell in Rust allows us to quickly deploy a bundled application without the need for compiling python into a binary, while still keeping the flexibility of python through module implementation.

Task 2 (Implement Assignment Creation): The assignment creation module creates the assignment description as well as the corresponding file structure. The assignment description contains the title, any assignment content, the course, due date, allowed attempts, and auto test configuration. The file structure for an assignment contains a submissions directory, and an auto test directory. Within the submissions directory, a directory for each student in the class must be made. The students in a class are found within the students.json file that is located in the class directory itself. This sets up a structure that will be used by later modules to access and update this information.

Task 3 (Implement Assignment Submission): The assignment submission module was created to submit any number of files to a submission. This module takes as keyword arguments a user, class name, assignment name, a programming language, and some number of files. Any arguments not provided through the command line arguments are then prompted for by the application. The module then does error checking to ensure that the inputted class exists, that the user is assigned to that class, and that the assignment is a valid assignment within the class. If this error checking is successful, the submitted files will be placed within the un-graded directory for the particular user in the corresponding class and assignment. Finally, the submission module also builds a json containing information that describes a submission. This includes the timestamp of when the assignment was submitted, that language it was submitted in, and the number of files that were submitted. Once the submission is graded, this json description will also contain the score that the submission received.

Task 4 (Implement Assignment View): The assignment view module displays the contents of an assignment description to the user. When an assignment is created, its details are saved as a json file. However, this json file is not a user-friendly way to display this information, so this module parses the json file into a format that can be presented to the user. Additionally, this module, much like the previous modules, takes in keyword arguments from the command line, or can take in the parameters by prompting the user. It then will find the corresponding class and assignment, if it exists, parse the json, then display the description.

Discussion of member contribution:

Charlie: My primary contribution to this milestone was the implementation of the assignment submission module. I started with a very simple python script that just took a file and copied it

to the corresponding directory. I then built it out to work with keyword arguments, partial keyword arguments and user prompted information. My next step was to make the code robust with error checking, as we want to avoid the modules ever crashing in the real world. Finally I set up the json description file, which will become useful when we implement the auto testing portion of the project. Besides this module, I worked with the group to help create the file structure we were going to use, so that files would be stored not only in a meaningful way, but also consistently so that they work across modules.

Michael: My primary contribution to this milestone was the implementation of the shell application. I decided to choose Rust as the language of the shell in order to more closely align with the need to compile the program into an executable binary. While this is possible with Python, the language isn't built to be compiled. The shell at present takes in user input and executes our custom commands using keyword matching. Future contributions to the shell will expand on the shell's capacity to mimic a bash shell. Once the client-server interaction is complete, a user that uses a valid custom keyword, i.e. assignment_submit will make a request of the server to run the command rather than the command being run on the local machine. This is to streamline the process of containerization as well as the scalability of the product while keeping implementation details obfuscated from the student user.

Tommy: My primary contribution was making the assignment creation module and the assignment deletion module. I started this milestone by creating a class hierarchy for the commands themselves. This will create inheritance that will improve the code during later stages of development. Once the classes were created, I used them to make the assignment creation module. After this was working, I made the assignment deletion module, which works very similar to the creation module, but it deletes the description and corresponding file structure rather than creating them. In addition, to this contribution I also set up the create course and delete course modules so professors can create courses to store assignments.

Logan: My primary contribution to this milestone was to make the view module. I started by making a simple python script to parse a json file and create a readable output from it. This however only used one specific json file, so my next step was to add the functionality to find the assignment as well. Initially I did this through user prompts, but then added the keyword argument functionality as well.

Task Matrix for Milestone 3:

Task	Charlie	Logan	Michael	Tommy
Client-Server Implementation	15%	20%	50%	15%
File Transfer	50%	10%	30%	10%

Auto Testing	20%	20%	10%	50%
Feedback System	20%	50%	10%	20%

Discussion of Milestone 3 Tasks

Task 1 (Client-Server Implementation): Our primary task of milestone 3 is to split the shell application into two coexisting parts. When a valid client connects to the shell, they will be able to use the same commands that were implemented in milestone 2, but the commands will be executed on the server instead of the user's system. The output of the task will then be returned to the user.

Task 2 (File Transfer): Once the client server implementation is set up, we need to add file transfer to assignment submission. This is important as the files will be transferred from the client machine to the server. To do this, we need to have a process on the server that is able to accept the files and place them in an "incoming" directory that will house all the file transfers. Then a separate process will need to move the files into their appropriate directories. Once this task is completed, we will have a working submit server and can move on to making it more robust.

Task 3(Auto Testing): After receiving the submissions, and placing them in the correct directory, the server will auto test them once resources become available. This is an important distinction as we want to prioritize file submissions rather than auto testing. As the deadline approaches, we expect the number of submissions to increase. If we simply auto tested the files as we received them, students would not be able to submit in a timely manner. This task will include using the configurations set within the assignment creation to spin up a docker container. We will then place the student submitted files into the container, and test them with the corresponding test cases.

Task 4 (Feedback System): After a submission has been tested, the feedback system will display the score and associated feedback. To complete this task, we will need to make a new module that will display scores as well as professor provided feedback. This means that the module must be able to determine which test cases were violated, and what feedback should be shown in response. Additionally, this module will need to be able to determine which feedback is appropriate given the output of the student's code.

Dates of meetings with the client/advisor:

10/22/2024 at 11am

Client/Advisor feedback

Task 1 (Implement the Shell Application):

- Consider implementing command options or flags. This could allow reusability of shell commands, so that the user does not have to remember a lot of commands. For example, with the view command, an option could be used to specify viewing an assignment, submission, or directory.
- Make sure that the shell commands that you are implementing are POSIX compliant. Ensuring standardization will improve usability.
- If the project can be in a running state by March, we should try to have a test run with a real assignment in my class. This would allow you to test performance, load balancing, as well as getting feedback from the students.

Task 2 (Implement Assignment Creation):

- Remove spaces from directory names: i.e. “Michael_Komar” vs “Michael Komar”. In general this will make the paths easier to create and navigate, especially if they are going to be exported by the professor.

Task 3 (Implement Assignment Submission):

- Timestamp the submission from the moment the user requests to upload a file on the user end. You don’t want to punish the student for the server not having enough resources to receive their submission.

Task 4 (Implement Assignment View):

- No feedback provided

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