Documentation

Documentation for the project codebase is available here:
https://harrismcc.github.io/classroom-voter/

Sprint Report

Ishaan: My contributions for this sprint focussed on the professor UI. I wrote a program to let professors describe polls. The poll is serialized and then sent over the wire. I also set up CI for our repository to automatically run tests upon committing. The most challenging part of the work was piecing it all together. There were a lot of changes to common data representations we were using, and when somebody changes that, everything breaks. I would estimate I spent 4 hours on coding, and 2 hours on meetings. I finished all the backlog items. I think we can continue to work similarly in the future.

Jay: implemented student-side UI, poll receiving and answering. Wrote up a design document and maintained the task backlog. ~7 hours? This should not have taken nearly as long as it did but between needing to familiarize myself with the codebase, wrestling git, and having to rewrite the code once, it did.

Douglas: My contributions for this sprint focused around the server. I spent most of my time building the infrastructure to connect the professor and students. To facilitate communication between professor and students I created a packet format for communicating the intended endpoint of the message. After creating the packet format I focused on handling multiple connections simultaneously using threads and storing all of the active connections in a list. Once the server could handle multiple connections I moved to receiving and forwarding packets to the intended location. Currently, the server can: receive polls from the professor and forward those messages to the students, receive responses from students and store them, and receive an result aggregation request and send off the results to the professor.

Learning the code base and implementing the server took between 6-8 hours. My productivity for this sprint was fine, it took some time to understand the code other people committed, but I was able to effectively use the data structures that were put together to accomplish the goal of setting up communication between the professor and students.

Harris: For this sprint nearly all of my work was on creating the base classes for the different types of objects we needed. This means a Poll class which represents the poll itself, as well as classes that are contained within it. This includes PollQuestion and PollRespose. Within the PollQuestion class, there are several child classes for different types of questions (such as multiple choice). Although there were no implementations of questions other than free response in this alpha, some groundwork was required to set up these other types for the future. After setting up these classes, I implemented methods and classmethods which allow each of them to be both converted and constructed to/from dictionary, json, and byte formats. This allows easy transmission of these objects over the network. After other sections of the code were implemented such as client, server, and professor, I spent the rest of the sprint debugging and tuning these conversions. This was exactly the work that was planned for this sprint, and it was completed on time.

Overall it took about 8 hours of work total for all of these parts. I was overall very productive and was able to accomplish everything early enough to take some time to understand what other team members had written in order to consider next steps for both this section and the project as a whole.

Professor UI - specify question, answer, anonymity level. Create new poll object

- Free response right now
- Read

Voter UI - Take in a poll object and render it. Let voter specify answer, update poll object

Objects - implement function/ update for server design

Server

- Schemas:
  - {
    - “endpoint”: string (either Announce_poll, Get_poll, …)
    - “Arguments”: {
      - ‘poll’: Poll object. Announce_poll MUST contain this key/value pair
    - }
  - }
- Poll object:
  - {
    - ‘response’: string representing the response, if available
    - ‘question’: string representing the question, if available
  - }
- Endpoints:
  - Announce_poll: Take in a professor’s poll, and store it
  - Get_poll: The voter asks for a poll, and the server sends the most recent poll to that voter
  - Get_aggregation: When a professor calls get_aggregation, an aggregation of all the poll results is sent
  - Get_response: the voter sends a response object with a response to the most recent poll

Harris

- Objects

Ishaan

- Professor UI

Douglas
- Server

Jay
- Voter UI