Comment Anywhere Progress Report

February 13, 2023 - Week 4

Luke Bates

On February 6th, we discussed the five implementation pipelines to focus on. We decided to focus on logging in, logging out, editing your profile's blurb, editing your account's email address, and changing your password.

On February 7th, Karl created a command line interface (CLI) and logger utility to the server backend. The server and CLI run within separate threads, which takes full advantage of the Golang programming language's concurrency features. This CLI is used to view information regarding the server, such as system logs and the number of active users. The CLI is also capable of properly shutting down the server.

Karl also implemented two logger middleware functions: One for logging a request when it first comes in, and another for logging a request just before it is done. These routines cause the console to display when logging is active, which is set by the environment variables, and can be altered at runtime via the CLI.

On February 9th, Luke and Frank collaborated in preparing the frontend for the upcoming integration test. Luke created the basic structure for the CafeOwnProfileDisplay class, which was the prerequisite for implementing ChangeProfileBlurb and ChangeEmail, while Frank focused on configuring the login button to communicate with the server. Later that night, Luke and Frank both submitted a pull request to the Comment Anywhere frontend repository containing their contributions.

On February 10th, Karl discovered an issue with the original idea to use cookies as the primary means of user authentication. The proposed method was to send a cookie, having contained an encrypted token, to the client. The server would receive this cookie from the client upon connection, decrypt the token, and send back a response indicating whether they have been authorized or not. However, this means of transporting cookies only applies to web pages hosted by the same domain. In our case, our clients are sending requests from a static browser extension.

The first solution was to utilize Cross-Origin Resource Sharing (CORS). This is a technology that utilizes HTTP headers to allow requests from outside domains. The idea was to enable CORS to allow cookies to be sent and received from any clients' browser extension. However, it was necessary for us to allow resource sharing from any possible domain. This option prevents us from using cookies, as there is a set rule stating that cookies may only be used for explicitly defined domains. Without this rule, there could result in Cross-Site Request Forgery (XSRF) attacks.

Karl's solution was to find an alternative to using cookies. Instead, tokens would be sent to the client via the ServerResponse array. The frontend's Fetcher is responsible for writing its token as a custom header to its requests.

On February 12th, Luke and Karl developed the CafeSection class. This was an abstraction for a container element, holding many child HTML elements. Everything, except for the collapse and expand button, would be hidden by default. Clicking the toggle button would hide or show the contents of the container, including various input boxes and text fields. As of now, this feature is used exclusively by the CafeSettingsWindow class. We both also decided to layout the password reset code on the same page as the new password field.

There had been few bugs present while testing the newly added settings window. One instance was showing the incorrect window upon entering the settings window. This was fixed by tweaking the backend to provide an email upon entering the settings window. Another bug was the email not being verified when the user changed it. This was contrary to the registration window, where email was verified to contain an "at" symbol, and a domain. The only fix was to apply that verification function to the input from the settings window as well. Other errors simply called for tweaks to the backend communication methods, such as when an email was changed and successfully updated in the database, it would show up as an error on the frontend anyways. There was also an issue where the profile blurb would be updated, but the data would not be refreshed until the user left the settings window and came back.

Throughout this week, we had to refactor some components of the design document. This was necessary to achieve a more feasible and efficient product, and also out of pure necessity. These include the inability to use cookies for authentication, merging the “password reset code” and “set new password” pipelines, and adding a parameter to the Fetcher to determine the HTTP request method.