Comment Anywhere

Week 2 Report, 1/30/2023

Karl Miller

On January 23rd, the Front End repository was created at github.com/comment-anything/ca-front-end. A basic directory structure was set up, referencing the prototype. A src folder contains typescript files. An assets folder contains html files, image files, .css (styling) files, and the manifest.json, which tells browsers how to treat the extension. A docs folder contains various documents in markdown format for our reference. Likely, the front end will follow a similar build process to the prototype. The typescript will "transpile" to javascript and the javascript will be "webpack"ed into a single script or two and bundled with the other assets.

We did not continue to develop the front end, because we decided to spend some time investigating implementation alternatives. While our prototype worked, it uses vanilla JavaScript to populate the DOM. This is very performant, but we have to manually assign CSS class names and style everything by hand. There are several front-end frameworks available, such as React, Vue, and Angular. Luke and Frank decided to investigate front end frameworks and see if any of them are a good fit for our project. We are especially looking for frameworks that easily integrate with libraries that provide a lot of predefined styling, so that we can have the sleekest buttons, text inputs, and layouts possible. Ultimately, as long as the framework can "pack" to a single javascript file at the end of the day, any one of them should be viable, though they may require some small changes to the design.

Luke also investigated Vite, a testing framework. During our prototyping phase, we did not investigate much unit testing. Luke looked at Jest first, but found that Vite was more streamlined. We will likely use that for front end tests.

I created the repository for the back end at github.com/comment-anything/ca-back-end on January 24th. Note that our repositories are set to private, and are all rights reserved licensing for now, but we are happy to share with Professors which can be accomplished via invitation to our github organization. Tests are actually very easy in Go. You define a function starting with the word “Test<name>” where name is the function you are testing. Tests exist in the same package as the functions they are testing. They take one parameter, a pointer to a testing.T. There is also a benchmark function, which takes a pointer to a testing.B. Running go test will execute the tests.

On January 27th, I spent time on the Makefile. To test server endpoint handlers, we will want the server to access a testing database. The testing database will run in the same postgres docker container as the production database and use the same schema. It will, however, be populated with test data. I created make commands to initialize and populate this database. A flag in the .env file will indicate whether we are in development or production mode, which will determine which database the server interfaces with.

On January 28th and 29th, I worked on tests for the back end. I created comprehensive unit tests for config.go, which sets the environment variables from a .env files which are used to configure, for example, ports and database access credentials. I created comprehensive unit tests for the server initialization and started unit tests for the first endpoint handler, the “register” handler. I discovered a go test command which will provide detailed coverage reports, including which lines of code are untested. We currently have 100% converage on the back end. I created detailed documentation for the makefile commands relating to the database and building docker

containers, including how to use the Postgres GUI, pgAdmin, to access the docker container's database.