

Urania

Purpose

The purpose of this document is to provide evidence for the different roles of the team members.

Introduction

This project is aimed to design a cloud-based snippet system utilizing AWS, Amazon Web Services. In this system, a user is able to create a snippet of code and multiple users are able to view the same snippet while new comments and updated contents appear.

In the system, there are three types of users:

1. *Administrator* - has special access privileges to manage the entire set of snippets (to delete a particular snippet, or to delete the stale snippets that are a predefined number of days old).
2. *Creator* - user which creates a snippet and is the only one who is allowed to modify the information about the snippet they created.
3. *Viewer* - any user who views a snippet created by another user. Both creators and viewers are able to edit the snippet text area.

In the following sections, the detailed information about the project is presented. The rest of the document is organized as follows: first, the team organization, members, and their responsibilities are described. Then, the team working design process is depicted. In the next chapter, the tools used during this project are outlined. Later, the accomplishments of our team, the reflection on the work, and the lessons learned are represented.

Team organization, members, and responsibilities

Team Members:

- Noel Qiao - Backend (AWS)
- Joseph Yuen - Frontend (React)
- Vladimir Vakhter - Backend (AWS)
- Shine Lin Thant - Frontend (React)

Even though team members interacted with most aspects of the program, the team was organized into 2 sub-teams: (1) those who mainly worked on the frontend (React) and (2) those who mainly worked on the backend (RDS, API Gateway, Lambda). We decided to split up in this way as we were initially interested in these domains and wanted to have a strong start to the project.

Initially, the front-end subteam included Joseph Yuen and Shine Lin Thant, and the backend team included Noel Qiao and Vladimir Vakhter. When we started to work on the admin view, subteams switched their roles to gain a holistic view of the system. As a result, each team member gained experience working with both the frontend and backend systems. In addition, all team members tested system functions through Unit testing and interacting with the UI.

CS509 Team Project Report

Our team typically met at least 4 times a week. The general structure of meetings consisted of a brief review of what was done with each team member explaining their accomplishments and areas where they ran into trouble. Some meetings were work sessions where team members would be available to help each other in their tasks if needed. During some work meetings, one or two members of the team would share their screen and perform the actual coding, while other team members watched, made suggestions, and looked out for errors. Before the end of the meetings, a discussion would always be held to review the schedule and to assign next tasks to each team member.

Process

We split the project up by use cases and we followed a “sandwich” style process. This means that we had a subteam working on the backend sections of a use case while the other worked on the frontend section. Before splitting up, we discussed the inputs and outputs that each team expected to make sure we were on the same page. Once both teams felt that their sections were completed, then we came together to hook everything up. This mainly involved working with the API calls and testing to make sure the use cases were working as expected. Occasionally, we would realize that changes needed to be made in the frontend or backend. In those cases, depending on the size of the change, we would either split up again or work together. We met often to check-in with each other even before we were ready to start combining the sections to make sure both teams were making good progress. If either team had any issues, we would jump in to try and help.

We utilized pair programming and individual programming to complete our tasks. The AWS tasks were split up by use case and could be more easily done individually. Since each use case corresponded to a Lambda Handler function and was linked directly to a method in the API Gateway, individual task assignments worked well. Frontend tasks were broken up into elements as well for each use case, though there was much more overlap in terms of how the code worked together.

To manage the quality of our code, we used JUnit test cases to test our models, DAOs and Lambda Handler functions. In addition to unit testing, we also tested our code in every step of the AWS system using the Lambda tests as well as API tests. Lastly, we made sure to test the use cases in the actual application to see if there were any glaring issues. Before each of us made changes to the code repository and made new releases, we tested our changes in application using localhost. We did our best to only push changes once we tested our sections and felt that no new major issues came up.

For meetings, we used voice and video calls for ease of communication. To quickly check-in with each other on the status of work or to have questions answered between meetings, we used Slack for texting. We used Clickup for project management so that all team members could go to one location to check on the current set of tasks.

Tools

Name	Purpose	Description
Eclipse	Programming IDE	A convenient coding environment, extensible and customizable by means of plugins, such as AWS Toolkit, EclEmma, or web development plugins
AWS	Handle persistent storage (RDS, S3) and process functions in the cloud (Lambda, API Gateway)	On-demand 'pay-as-you-go' cloud computing platform that provides multiple services, such as RDS (relational database that stores the entities, snippets and comments, presented on the system), API Gateway (maintaining REST API and managing HTTP requests/responses), S3 (cloud storage for hosting the designed system), and Lambda (serverless compute for implementing the controllers for the use cases)
React	Javascript library for building UIs	Accelerated and simplified the frontend development (interactive user interfaces)
NPM	Node Package Manager	Managed the dependencies for the JavaScript programming language. Used for testing the application locally (npm start) before uploading the production version (npm run build) to S3
Swagger	Create REST API	Simplified the design of REST-based API calls before importing them into AWS API Gateway in the form of yaml files
Sketch	Design prototype mockups	Used to draw storyboards for the GUI that displayed how the different types of users interact with the system
Google Suite	Meeting Minutes and Documentation	Synchronous real-time development of documentation
Git	Handle code versioning and storage	Tracked changes in the source code files, coordinated work among the team members, stored the source code files
Slack	Asynchronous communication	A channel for sharing thoughts and materials outside of the regular in-person meetings

CS509 Team Project Report

Zoom	Synchronous communication	Used for working sessions for discussions, assigning tasks, and collaborative team work
Clickup	Project management	Coordinated the team work
MySQLWorkbench	Visual Database Design Tool	Simplified work with SQL (create tables, execute scripts with SQL requests) , visualized the current state of the system (snippets and comments)

Best Tools

We thought that Git, Swagger, Slack, and MySQLWorkbench were extremely helpful tools. They were reliable and were relatively straightforward to use. Git specifically was helpful as it allowed us to revert to old code if a new version was broken. Swagger was useful as it flagged errors and listed existing functions when writing the API. Slack provided a simple way to share documents and communicate. Lastly MySQLWorkbench allowed for team members with less experience with SQL to make rows and easily see the state of the RDS.

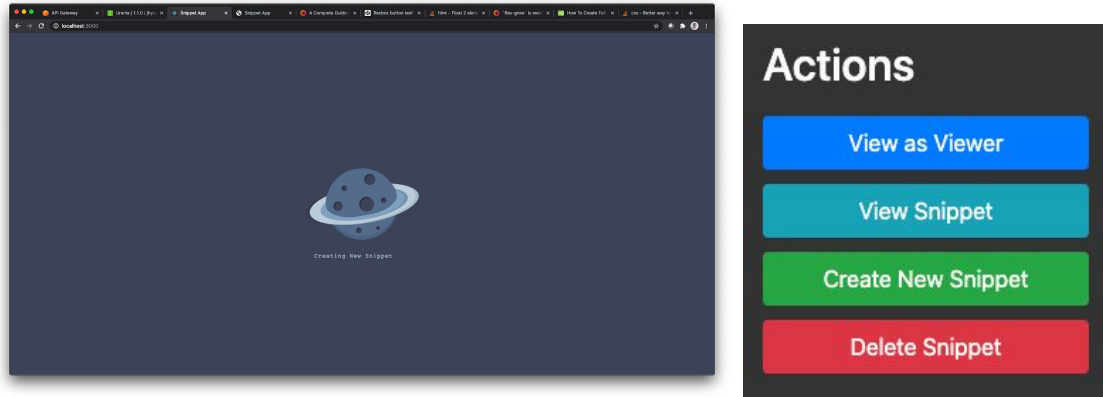
Worst Tools

Even though Eclipse and ClickUp are useful tools, they weren't as valuable to the team compared to other tools. Eclipse sometimes had issues when issuing Git commands. The history tab would not always update with the latest versions of the code which was frustrating. ClickUp wasn't as useful to the team as it is built for larger Agile teams. Our team met so frequently and was relatively small, so we stopped using ClickUp after the first iteration. Instead we just added assigned tasks to our minutes.

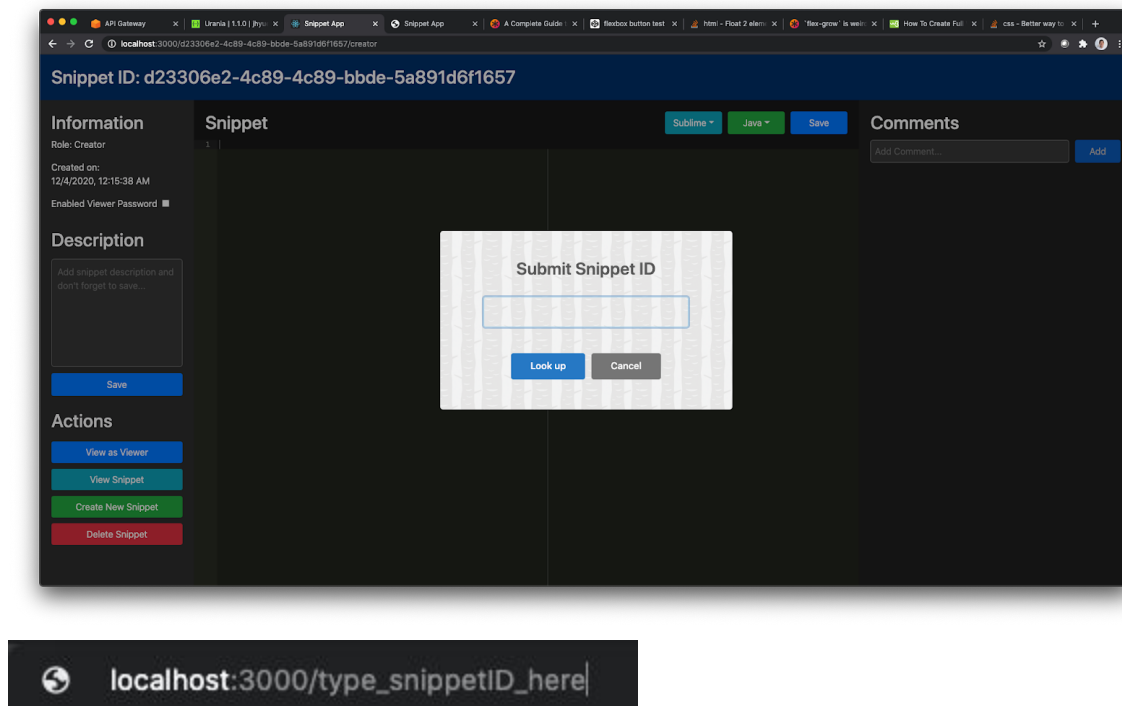
Accomplishments

The team developed a React AWS application that allows users to create, edit, delete, and comment on code snippets. We also created an admin page for admins to manage snippets and delete stale snippets. The following subsections list the required use cases and their respective implementations in the application.

Create Snippet



View Snippet



Update Snippet Info

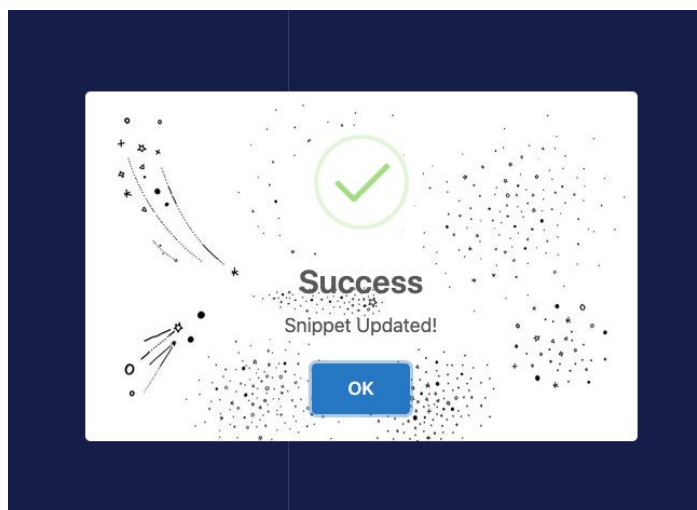
<h4>Information</h4> <p>Role: Creator</p> <p>Created on: 12/4/2020, 12:15:38 AM</p> <p>Enabled Viewer Password <input type="checkbox"/></p> <h4>Description</h4> <div>Snippet info goes here.</div> <div>Save</div>	<h4>Information</h4> <p>Role: Viewer</p> <p>Created on: 12/4/2020, 12:15:38 AM</p> <h4>Description</h4> <div>Snippet info goes here.</div>
---	--

Update Snippet Text

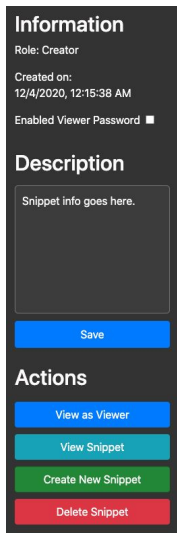
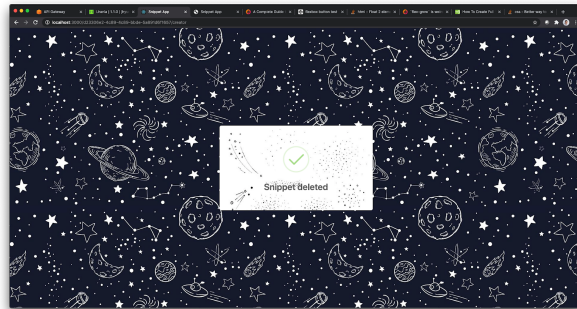
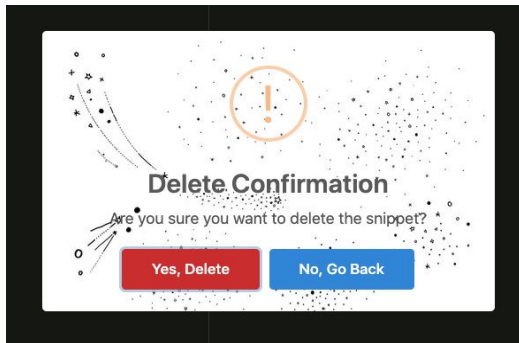
Snippet

Sublime ▾ Java ▾ Save

1 Snippet text goes here



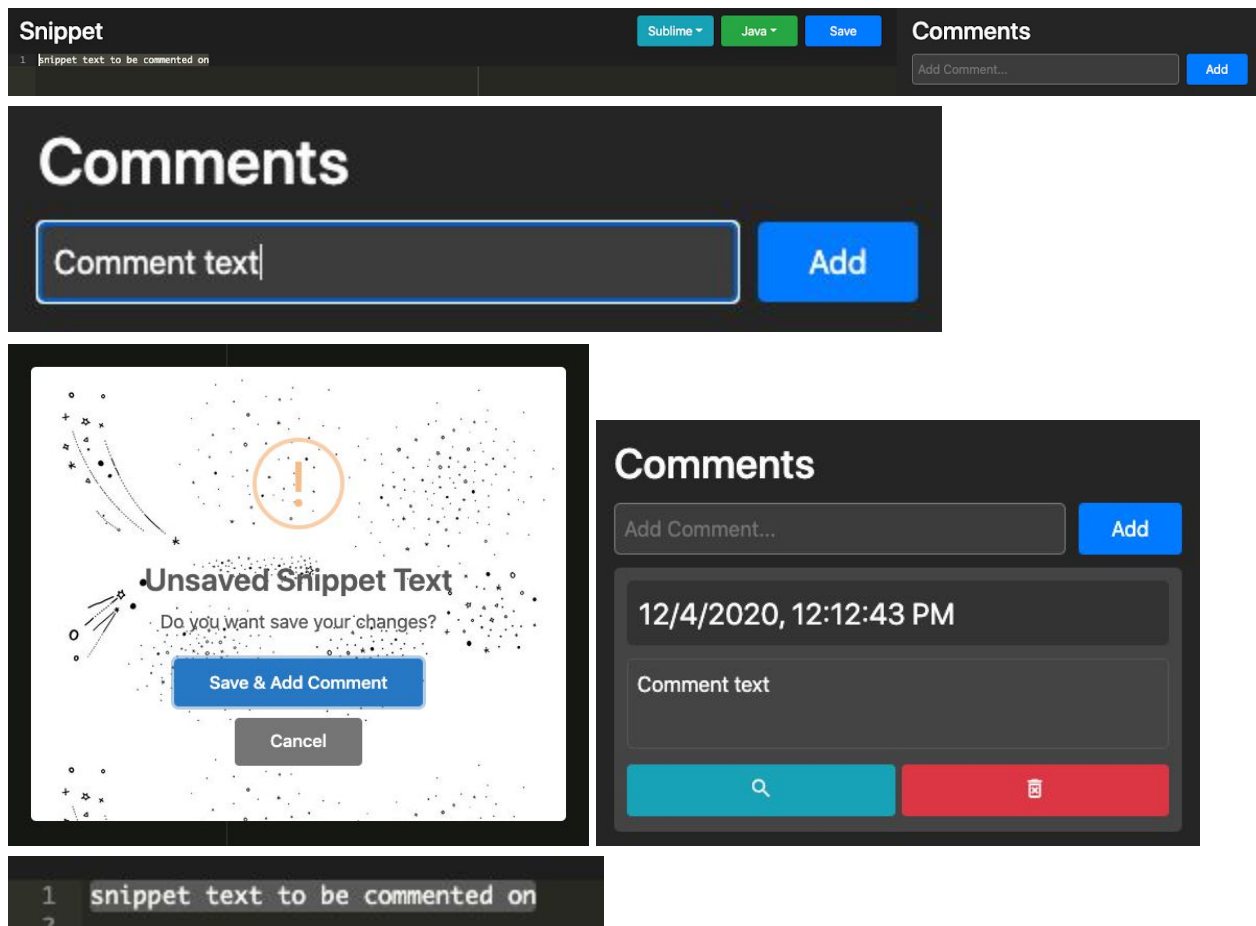
Delete Snippet as Creator



Delete Snippet Observed as Viewer



Create Comment

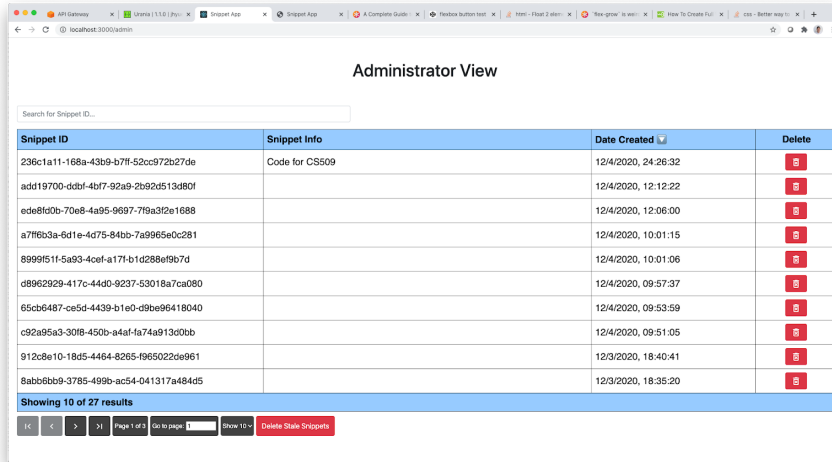


Delete Comment



CS509 Team Project Report

List Snippets



The screenshot shows the 'Administrator View' of a web application. At the top, there is a search bar labeled 'Search for Snippet ID...'. Below it is a table with four columns: 'Snippet ID', 'Snippet Info', 'Date Created', and 'Delete'. The table contains 10 rows of data. At the bottom, there is a pagination bar showing 'Showing 10 of 27 results' and a 'Delete Stale Snippets' button.

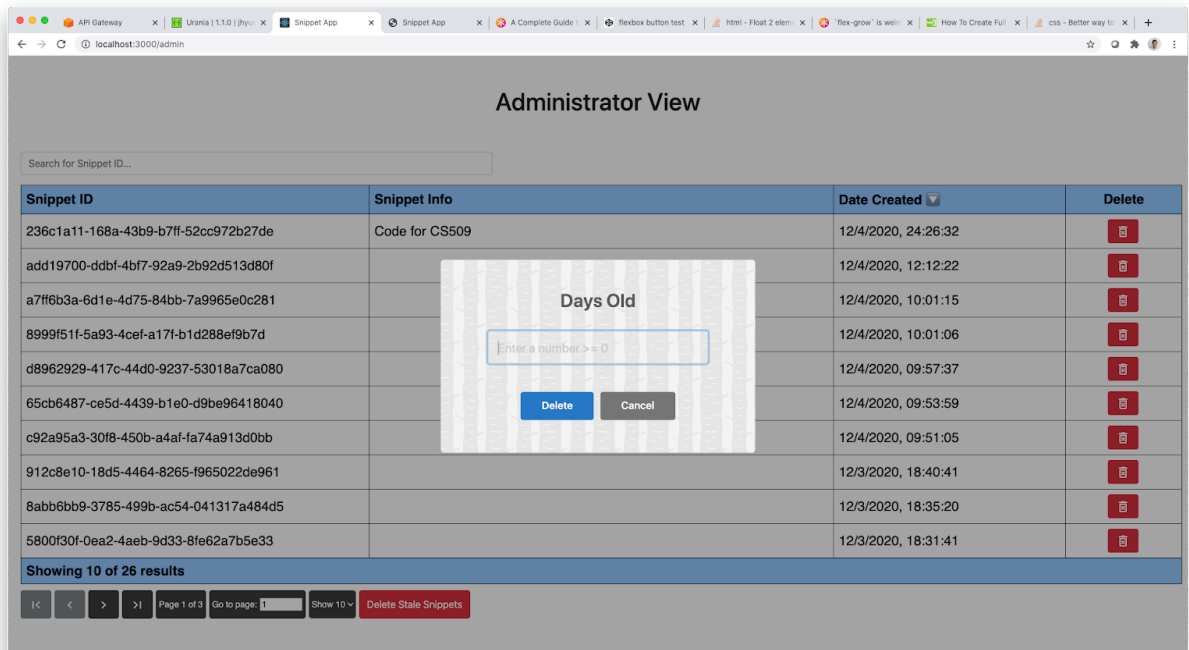
Snippet ID	Snippet Info	Date Created	Delete
236c1a11-168a-43b9-b7ff-52cc972b27de	Code for CS509	12/4/2020, 24:26:32	
add19700-dbf4bf7-92a9-2b92d513d80f		12/4/2020, 12:12:22	
ede8f0b-70e8-4a95-9697-7f9a32e1688		12/4/2020, 12:06:00	
a7ff6b3a-6d1e-4d75-84bb-7a9965e0c281		12/4/2020, 10:01:15	
8999f51f-5a93-4cef-a17f-b1d288ef9b7d		12/4/2020, 10:01:06	
d8962929-417c-44d0-9237-53018a7ca080		12/4/2020, 09:57:37	
65cb6487-ce5d-4439-b1e0-d9be96418040		12/4/2020, 09:53:59	
c92a95a3-30f8-450b-a4af-fa74a913d0bb		12/4/2020, 09:51:05	
912c8e10-18d5-4464-8265-f965022de961		12/3/2020, 18:40:41	
8abb6bb9-3785-499b-ac54-041317a484d5		12/3/2020, 18:35:20	

Delete Snippet as Admin

Delete



Remove Stale Snippets



The screenshot shows the 'Administrator View' of a web application. A modal dialog box titled 'Days Old' is displayed in the center. The dialog has a text input field labeled 'Enter a number >= 0' and two buttons: 'Delete' and 'Cancel'. The background table is dimmed, showing the same list of snippets as the previous screenshot.

Snippet ID	Snippet Info	Date Created	Delete
236c1a11-168a-43b9-b7ff-52cc972b27de	Code for CS509	12/4/2020, 24:26:32	
add19700-dbf4bf7-92a9-2b92d513d80f		12/4/2020, 12:12:22	
a7ff6b3a-6d1e-4d75-84bb-7a9965e0c281		12/4/2020, 10:01:15	
8999f51f-5a93-4cef-a17f-b1d288ef9b7d		12/4/2020, 10:01:06	
d8962929-417c-44d0-9237-53018a7ca080		12/4/2020, 09:57:37	
65cb6487-ce5d-4439-b1e0-d9be96418040		12/4/2020, 09:53:59	
c92a95a3-30f8-450b-a4af-fa74a913d0bb		12/4/2020, 09:51:05	
912c8e10-18d5-4464-8265-f965022de961		12/3/2020, 18:40:41	
8abb6bb9-3785-499b-ac54-041317a484d5		12/3/2020, 18:35:20	
5800f30f-0ea2-4aeb-9d33-8fe62a7b5e33		12/3/2020, 18:31:41	

Non-Required Accomplishments

We implemented an optional password requirement for snippets so that the snippet system offers an added layer of protection, and for creators to be able to limit their audience. We also added several text editor modes such as Vim to allow more flexibility for users. The most important feature that we implemented that was not required was allowing users to identify what snippet text was associated with a specific comment.

Incompleted Features

We intended to implement a dark mode for the admin screen since our main viewer screen has a dark mode aesthetic. We also attempted real-time browser communication (web sockets or Redux) to eliminate the need for manually refreshing upon updates such upon snippet deletion and comment addition. However, we did not have enough time to fully implement these additional features.

Deliverables

The main deliverable is the project which can be found at this link: <http://uraniasnippetssystem.s3-website.us-east-2.amazonaws.com/>. In addition to that, we created README files as an additional user manual to explain the different elements of our project as well as specific interactions that may be unclear at first.

Reflection

What worked, what didn't work

What Worked

At the beginning of the course, the professor recommended that we focus more on functionality than the aesthetics of the application. The reasoning was that a pretty UI may impress people initially but quickly disappoint once the application is used. We adhered to the professor's advice and chose to work on less of the UI in the first iteration. Although we had a very non-aesthetically pleasing site by the end of the first iteration, the time we would have spent improving the UI was spent laying the functional foundation of the program which made later iterations easier and faster to implement. When we eventually improved the UI, we could do so without the stress of wondering if our features worked.

What Didn't Work

For our small team, the project management approach using ClickUp did not work as well as we thought at the beginning. We wanted to use ClickUp as an easy way to communicate tasks that were completed, in the process of being completed, and needed to be completed. We found that since the project was broken up into manageable sized tasks and because we met frequently, it was much easier to communicate that information in team meetings. We also recorded that information in our team minutes so it became somewhat redundant.

Since we all have different course schedules and work loads, scheduling weekly meetings was difficult. We all had to be much more flexible in terms of scheduling meetings, and of task goals

CS509 Team Project Report

and deadlines. Luckily for us, we were all very flexible and we didn't run into many scheduling issues.

Our biggest mistake

The biggest mistake that we made was not understanding the current React design patterns well. Our React code utilized classes whereas newer React projects only utilize functions. Another mistake that we made regarding React was the lack of Redux. It would have been very helpful for us to use Redux to manage the state of our frontend application instead of using many callbacks.

If we had to change aspects of our project, we would try to implement the incompleting features as discussed earlier.

If we were to suggest changes for future classes with respect to the project, we would like to have implemented more design patterns learned in class into the final project.

Lessons learned

If you use React, functional components can provide more flexibility and functionality than class-based components. This is because functional components use React hooks, and a lot of the latest libraries, such as Redux, work only for hooks.

We managed to work well as a team without any issues. We want to recognize the previous lessons learned that we utilized to have a successful project. One of the lessons we used early on was to identify the strengths and weaknesses of each team member so that we could get off to a good start. Another was making sure everyone was on the same page in terms of motivation and understanding each other's expectations for the group project. We believe this helped prevent serious conflict amongst each other.

Advice to future teams

Meeting frequently is the most important advice that we have. It helps a lot in terms of supporting each other, knowing where different parts of the project are at different times, and making sure that all elements get completed. In addition, we felt we benefited a lot from splitting the project into frontend and backend so that we had "experts" in different parts of the project.

To get the most out of this class, teams should discuss feature implementation and teach each other about the implementation. When we did this, everyone on the team was able to get a good sense of how different elements of the project worked together even if not everyone did every part. We even took this step a bit further and role-swapped teams so that we could all do some coding in the front and back end. We were able to do this because we were ahead on the project and because the admin related use cases were a perfect point to swap.