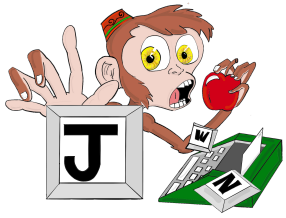
CS509 Team Report:
Team CLIO (aka Infinite Monkey)

Purpose

The purpose of this document is to provide in detail the project teams organization, roles, processes, tools, and accomplishments to deliver a successfully collaborative online text editor system. The team has also included a reflection and lessons learned section as this provides valuable insight into the iterative design processes when using new tools to develop a scalable and robust system.



Introduction

The goal of this project was to develop an online text editor that supports multiple user contributions. The collaborative nature of this editor allows users to edit and comment on the snippet of code while other users view these changes in real time. There are three actors that this system was designed for: creator, viewer/s, and administrator. The user that created the comment is called the creator of the code snippet. The creator has special privileges such as deleting the snippet, deleting comments, editing the snippet metadata, and also inherits the standard user (visitor) functionalities. Additionally, this snippet system has a password-protected administrator site that lists all snippets currently available. This administration site allows the admin user to delete individual snippets, delete snippets created before a specific date (stale snippets), and provides a hyperlink to each snippet.

The project specifications indicated that this snippet system had to be hosted using Amazon Web Services (AWS), use lambda functions for request handling, and use SQL for database storage. The Java lambda functions improved security and lowered browser requirements, as all traffic, requests, and storage was done on AWS servers. This report details how the team archived the design and implementation of the snippet system, along with notable challenges and solutions.

Team Organization, Members, and Responsibilities

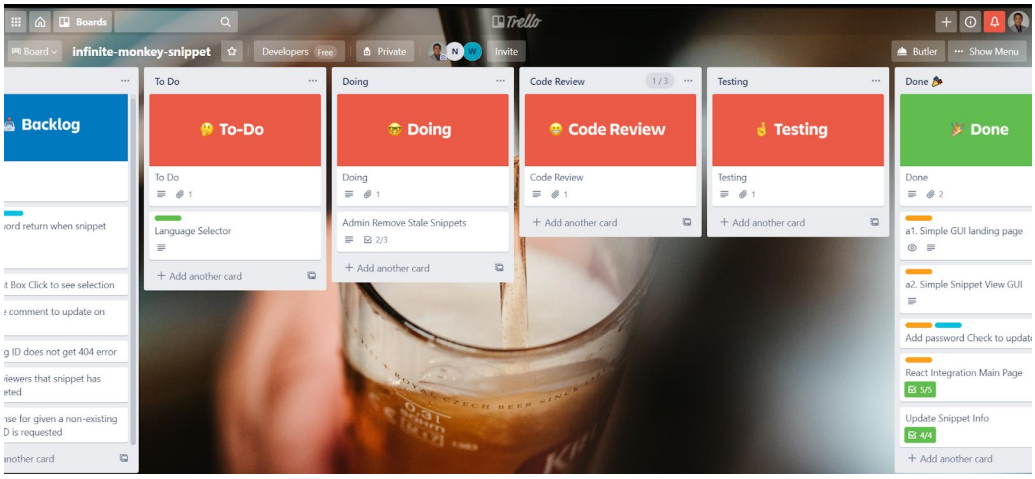
Our team was composed of three members: Nicholas Delli Carpini, Josue Contreras Albuja, and William Campbell. Due to the team's small size and similar programming skills we did not initially develop formal roles and responsibilities. This allowed each team member to familiarize themselves with the frameworks, libraries, and design patterns that would prove helpful later in the project development and each member's professional skills. However, roles naturally formed as the project continued, and individual team members became knowledgeable

on different aspects of the project. This natural development allowed the team to easily split tasks and perform better with every iteration of the project.

The team was able to manage a good balance of working and discussion meetings. Due to the small team size and the flexibility of each team member we scheduled two main meetings: one at the beginning and one at the end of each iteration/deliverable. Additional team sub meetings, working meetings, or discussion meetings where scheduled on a need basis, in between the two main meetings. To schedule meetings our team schedule times at the end of meetings to avoid filling out forms or having to check availability. For informal meetings our team communicated through slack. Most of the working meetings happened after team members worked on their assigned tasks, for code review purposes, and code merge.

Individual team members worked on tasks that were picked by each team member. There was no overall team leader as each team member took every week to check on the progress of each other. This was easy since there were only three team members. Either way the tasks were allocated by the project manager in the Trello to keep up to date. One sub-team formed between Nick and Josue as they took the lead in the GUI development and design. This was the case since a new JavaScript library, REACT, was used and team members had to learn how to use it. Then code reviews and merges where conducted by the whole team.

The organization of the project was done through the Trello application. This made it easy to track each task that individual team members were working on. Trello also allowed for each team member to view in what phase of development certain tasks where. Each card had a detailed or summarized description of a functional requirement, bug to fix, or extra functionality that our team wanted to add. The Trello was divided into the following columns: Backlog, To-Do, Doing, Code Review, Testing, and Done. Each card or task was placed in their respective column and each team member could view this when logging in to work. The following screenshot shows the layout described above. Github was also used to organize the code as it enabled the production code to be kept in the main branch while developing features and functionality in parallel. Pull requests and merges where overlooked by team members in meetings and tested to ensure proper and full functionality of the system.



Infinite Monkey Trello Layout

The following table demonstrated the roles that each team member developed throughout the design, implementation, and success of the collaborative text editor system. Each team member played a crucial role in the development of the system.

Team Member Name	Roles
Nicholas Delli Carpini	❖ GUI Designer/Developer ❖ AWS Developer/Lead ❖ REST API Developer ❖ Backend Developer ❖ SQL Database Lead ❖ Documentation Editor
Josue Contreras	❖ GUI Designer/Developer ❖ AWS Developer ❖ REST API Developer ❖ Backend Developer ❖ Documentation Editor ❖ Project Manager
Will Campbell	❖ Unit Test/Testing Lead ❖ AWS Manager ❖ Documentation Author/Editor ❖ REST API Developer ❖ Backend Developer

Process

Our process for getting our work done efficiently was an admittedly casual one, but still very effective, as we were able to consistently get our work done on time with little to no crunch time required. What would typically happen for each iteration is that we would first have a Zoom meeting as a group to discuss what will need to be worked on for this iteration and who might be able to handle these tasks. This often entailed walking through use cases together, as well as discussing how we expected functionality to play out in practice. We would sometimes begin work together in this Zoom call as well. From there, each member would typically work alone or with a partner on whatever tasks they were able to claim that weren't being worked on yet. We would look at the Trello board to see who was working on what, and if something still needed to be done that a member felt able to accomplish, they would claim the task and work on it alone. But help was also available as needed. If anyone ever encountered a problem they were confused about, they would send a message in the Slack, and other members would join them in a Zoom call to help get past the problem. Finally, as the due date of a given iteration approached, we

would schedule a few more group Zoom calls to finish up any remaining work together, as well as complete the write-up and prepare our full submission.

This was the process we followed for each of the iterations; the only change really that ever happened was the inclusion of the Trello board after the very first iteration. We didn't feel the need to make any adjustments to this process as the project progressed. The biggest reason for this was simply that it was working well for us. Each iteration came and went without any problems or difficulty raised by the way we worked, so we didn't feel any need to change it. We think this worked particularly for our group thanks to our group's small size and similar level of programming competence for each of us. With so few members, many things become easier that can be a challenge for larger groups. For example, it's easier for us to schedule meetings on short notice with fewer conflicting schedules. Also, we think the smaller group size helps keep members more inherently accountable for pulling their own weight; with fewer members, someone not doing as much work as is expected of them becomes more noticeable and problematic. We think this is why our process of claiming tasks as we each became available worked well. Another reason this worked well was thanks to our understanding of each other's coding abilities and knowledge. Some of us felt more comfortable or knowledgable with different areas, so we were able to let members simply claim tasks they felt comfortable working on, and in this way were able to finish each task efficiently.

The main techniques we used when working together were pair programming and use cases. Particularly with work on the front end, pair programming was frequently utilized to ensure fewer mistakes were made so we didn't have to go back to fix things later. This was especially important for us so that we could test back-end functionality without needing to worry that the front end was contributing to the problem. Use cases were also very important to us. We would use the use cases provided for that iteration to guide our development process, ensuring we completed expected tasks before getting distracted with interesting side features.

Requirements and code quality was managed via peer reviews in the Zoom calls leading up to iteration due dates. We would take time in these calls to review the code we were going to submit as a group and make adjustments as needed. For test cases, our method was to ensure that each handler was tested thoroughly by creating a test case for each possible status code that could be returned (with the exception of a couple of cases where intentionally getting certain error codes would have been especially difficult). Thanks to this process, we were able to be sure that our code was working as intended, and reach code coverage above the expected amount.

Our primary means of communication were Slack and Zoom. Slack was definitely used the most, being the place we would post questions or problems we had, notify others of what we were each working on, and schedule Zoom meetings. In our Zoom meetings, we would be able to more thoroughly communicate what problems we may have run into and work on solving them together, or simply coordinate work that required us all to be there, like the group write-ups. Thanks to our use of Slack and Zoom, we were able to keep very open communication so that everyone felt supported while working separately.

Accomplishments

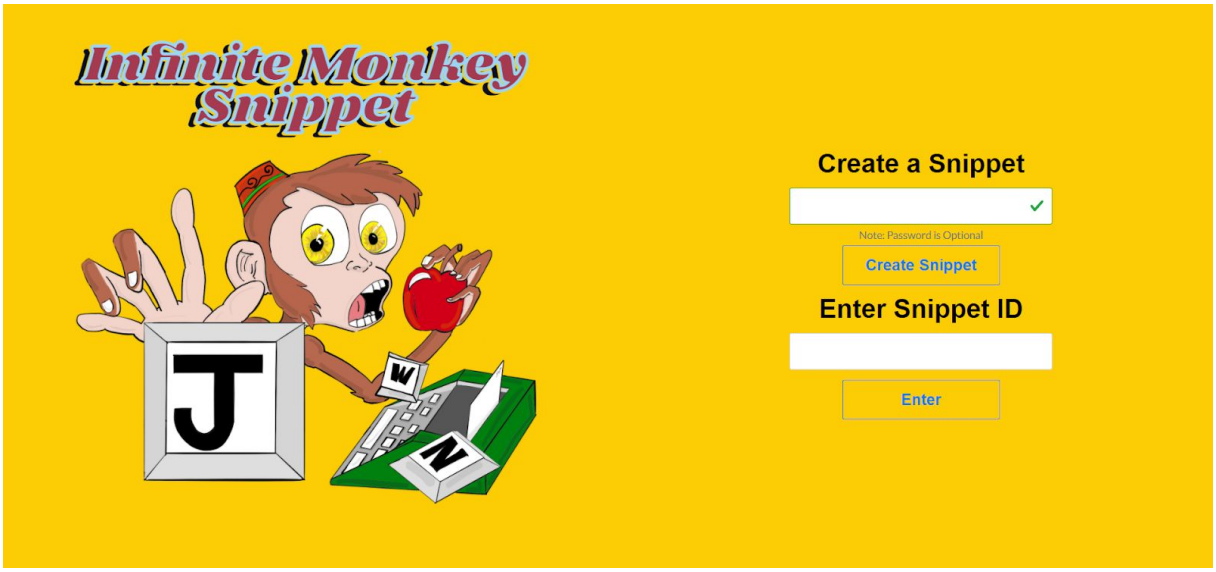
By the end of our time working on the project, we were able to implement all of the required use cases! These use cases and screenshots of the final GUI are outlined below.

Use Cases Golden Accomplished

Use Case	Actor	Entry Criteria	Exit Criteria	Event Flow
Create Snippet	A Creator	None	New Snippet created	1. Creator requests new Snippet 2. System creates Snippet and updates display to empty Snippet
View Snippet	A Viewer	The Snippet exists	Viewer now sees Snippet	1. Viewer requests top view Snippet by ID 2. System updates display to requested Snippet and its comments
Update INFO	A Creator	Snippet was created by Creator	Snippet INFO is updated	1. Creator requests to update INFO 2. System updates Snippet INFO, any Viewers see this change
Update TEXT	A Viewer or Creator	Snippet exists	Snippet TEXT is updated	1. User requests to update Snippet TEXT 2. System updates TEXT and all viewers see this change
Delete Snippet	A Creator	Snippet was created by Creator	Snippet and its comments are removed	1. Creator requests to delete Snippet 2. System removes Snippet and redirects users to homepage
Create Comment	A Viewer or Creator	Snippet exists	A new Comment is created	1. User requests to create a Comment for the Snippet 2. System creates Comment, all viewers see this
Delete Comment	A Viewer or Creator	Snippet has at least one Comment	Desired Comment is removed	1. User requests to delete Comment 2. System removes Comment from Snippet, all Viewers see this
List Snippets	An Admin	None	None	1. Admin requests list of all Snippets 2. System shows all Snippets in reverse order of creation
Delete Snippet	An Admin	Snippet exists	Snippet is removed	1. Admin requests to delete Snippet by ID 2. System removes Snippet, all Viewers are redirected to homepage
Remove Stale Snippets	An Admin	None	Snippets older than N days are removed	1. Admin requests to delete Snippets older than N days 2. System presents a list of remaining Snippets

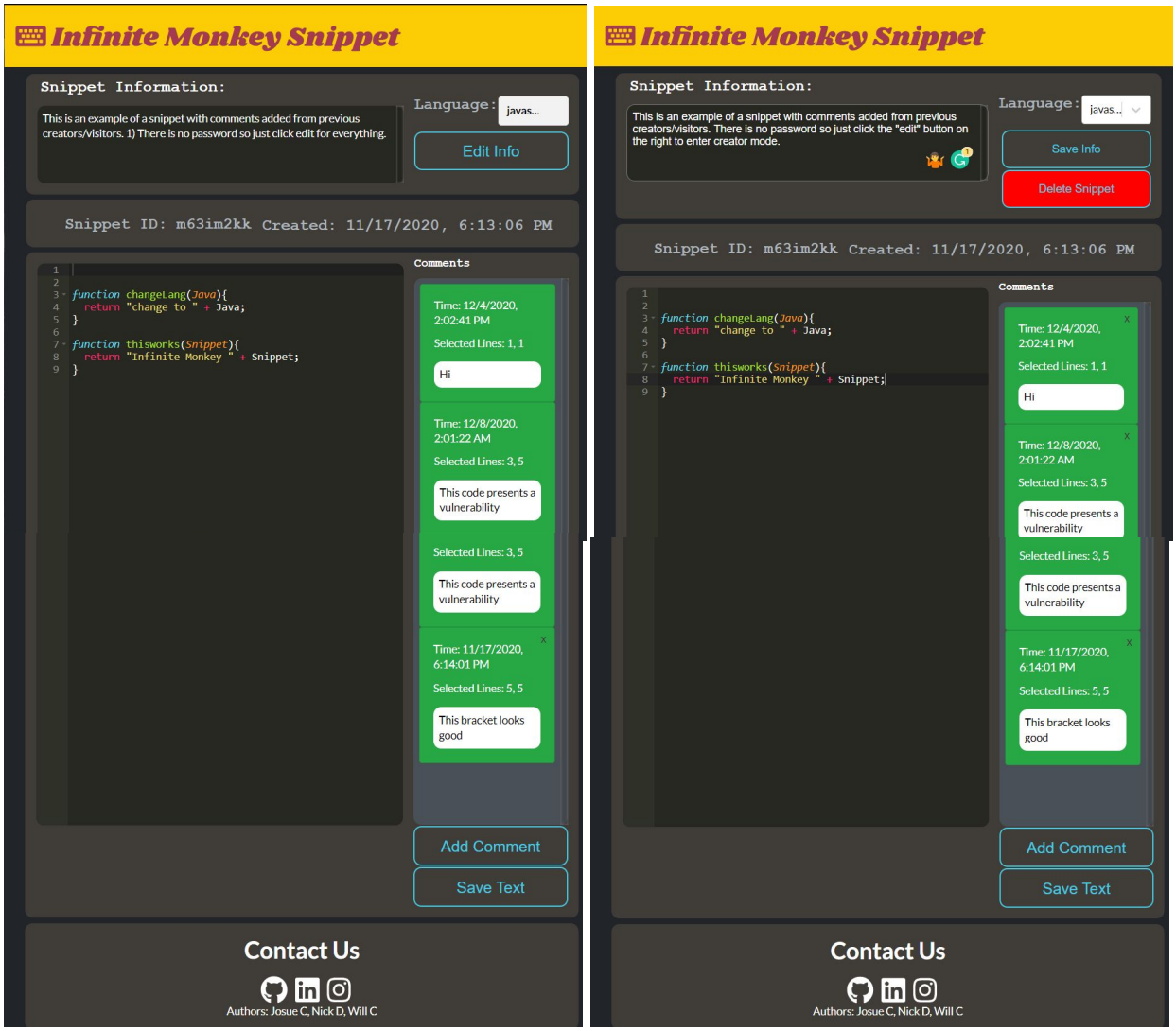
Snippet Site

The following screenshot shows the final home page for the snippet system. This page allows users to create a snippet with an optional password and go to an already existing snippet to view it. When a snippet is created a unique ID for the snippet is created and displayed in the next screenshots. The password allows creators to be given extra features on the snippet. It is important to note that the password for a snippet is not secure as any password with no restrictions/rules or encryption can be created. The password functionality only offers a way to separate visitors from creators and is in no way intended for high security reasons. The second functionality of this page is to enable visitors to view an existing snippet by a given unique snippet ID. When the unique ID is given this site redirects the viewer to the snippet with limited privileges.

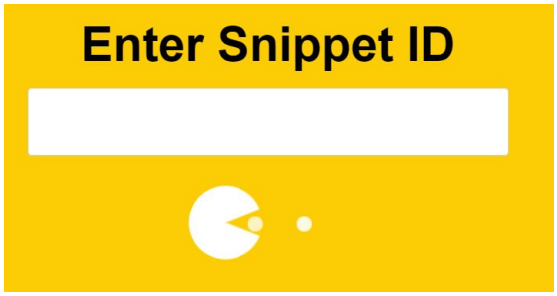


The following screenshot shows the main snippet page. This page contains most of the features and functionality from the use cases stated in the previous table. The left screen shot below shows the main snippet site with viewer restricted functionality. The right screen shot shows the snippet site but with the creator functionality enabled. For a normal viewer snippet site the functionality allowed is as follows. Snippet metadata is displayed on the top section. The middle section shows the snippet text and comments list. The Ace editor was used to display the snippet text as it provides some really nice features and it is a high performance web editor. The snippet text is highlighted depending on the programming language set by the creator. The auto complete feature is also enabled and assists the programmers. The comments section allows viewers and creators to add comments on selected text. These comments are displayed every n seconds (set by the programmers). A comment is added with the selection lines, timestamp displayed, and a textbox for the comment text. The two buttons allow the user to add or cancel a comment. The creator inherits the viewers functionality and is able to do the following added

functionality. The creator can edit the info, change the programming language, delete comments, and delete a snippet.

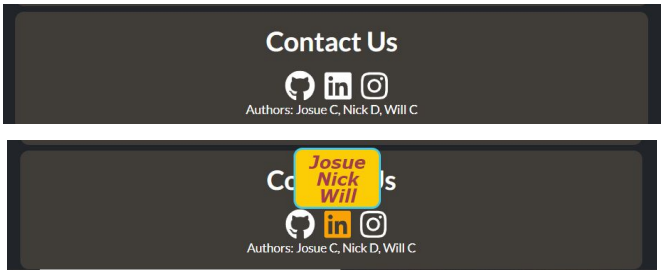


An extra golden nugget was added to the GUI that is the pac man loading symbol. This affirms users that a request was made after the action that they took. This is important since it allows them to visually see that a process is happening.



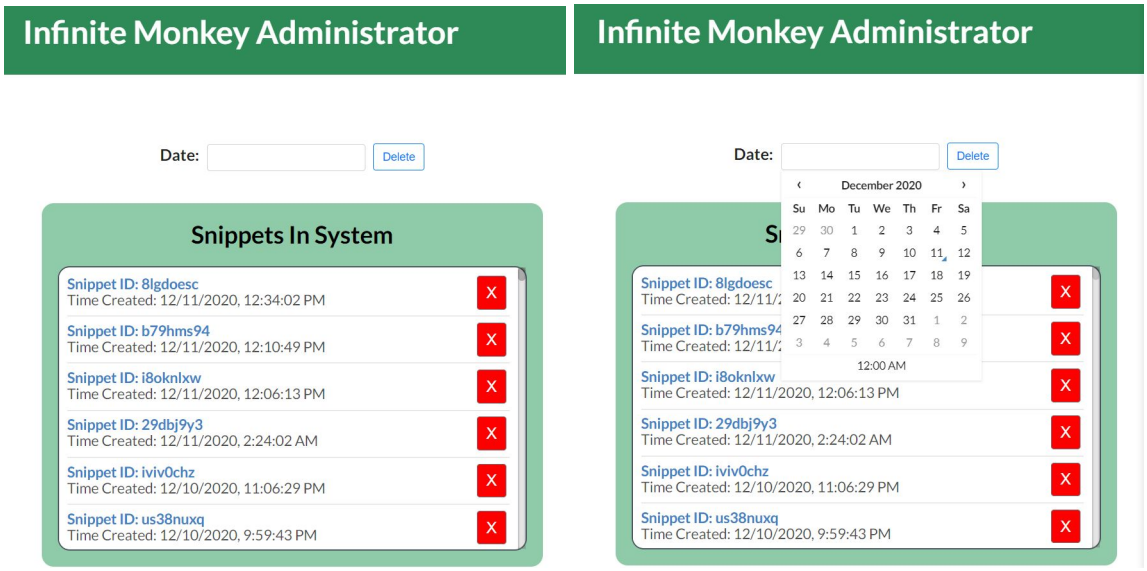
Contact Us Card

The additional “Contact Us” section to the snippet main site was added at the bottom of the site. This is important for our team as we aimed for the successes of the project. We were able to design and implement a working system that looks professional and our team is proud of. This contact section allows the visitors to hover over each contact icon and see the names with hyperlinks of each team member's social sites. This will allow possible employees and other developers contact the team members if interested in the project developer.



Admin Site

The project specifications also required an admin system. The following screenshots show the GUI with all the admin functionality requested by the use cases. With this site the admin is able to see a list of all the snippets in the database with their respective unique ID and time created. The admin is able to delete a single snippet or all snippets older than N days by picking a date in the dropdown calendar and pressing the delete button. The admin can also view the snippets by clicking on each snippet ID has a hyperlink to the snippet main site for that ID. The team decided to keep a simple GUI design for this site so the admin can easily perform tasks without having additional overhead of distraction.



Incompleted Features

There were three main additional features that our team was unable to complete or finalize for this project.

The first one pertains to the snippet GUI comment creation. Our team wanted to enable the add comment button next to the snippet mouse selection. This would clean up the GUI and would allow users to easily add comments as they highlight texts in the text editor.

The second feature that our team wanted to add was to highlight text pertaining to the comment that was clicked on. Our team wanted to allow the user to click on the comment card and then the text editor would highlight the lines pertaining to that comment.

The third feature that the current site partly works is reactive GUI design and partial mobile compatibility. This is partially working; some components where moved that broke the mobile GUI parameters. This can be easily fixed, but the team was not able to get time to fix this extra feature.

Deliverables

The following table shows the deliverables that our team has provided for this project.

Deliverable	**Purpose**
User Manual	Provides step-by-step instructions to complete the various actions described by the use cases. Also provides links to the main and admin pages of our site.
Code Repository	Anyone who is interested will be able to observe our code to see how we went about implementing the functionality for our site.
Code Coverage	Shows that our JUnit test cases utilize an acceptable amount of our code when testing the full functionality of the site.
Git History	Provides a timeline of the various commits made to the repository, showcasing how things developed iteratively over time.

Reflection

What worked, What didn't work

To reiterate a bit from earlier, we think our work processes really worked well for us, despite the more casual nature of it. Each member was able to claim tasks they felt comfortable with, and work in time available to them. This was likely the case thanks to our small group size, and likely not a good way of handling things if our group was much larger. But thanks to our smaller group, we had more freedom to schedule meetings as needed and allocate tasks independently, and that all worked really well for us.

Our Biggest Mistake

Our biggest mistake was probably our decision to jump into the creation of the front-end without putting enough thought into the design process and what tools were available to us for it. We ended up creating a minimal front-end for early development that eventually needed to be rebuilt from scratch once we decided to start using REACT. This meant that a lot of time needed to be put into shifting things over to the new layout. This could have been avoided if we had spent more time before we got started to consider our options. We could have spent more time as a group looking into different options for making the site look better and more modern, and started the project with a layout that looked and performed better from the beginning.

Changes We Would Make

The only particular changes that come to mind would be to implement the features we were unable to finish in time. It would definitely be nice to have mobile compatibility to allow for work on the go through your phone or tablet. Also, having comments highlight the text they are referring to, similar to Google Docs, would definitely help with clarity, so that's definitely something we would include if we had the time.

Lessons Learned

Things We Learned

One important thing we learned was to thoroughly test our code every time new functionality or features were added. Early on, we had some cases where multiple bits of functionality were added at once, and some confusion arose during testing because of this; this was avoided in future iterations of course. We also learned the importance of reviewing our code together before submitting. This really helped ensure that everything was working properly and in a way that wouldn't make things more difficult down the road.

One lesson that we were aware of but reinforced over the course of this project was the need to keep the git repository clean, with a working 'main' branch at all times. This way, we can work on branches without fear of ruining what works, and make changes with the knowledge that we can return to a previous working commit if things failed.

Lastly are a couple of important lessons from writing the code of the project. The first is to start with a simple backbone of functionality before branching out into more complicated things. This way you can know for sure what works when attempting to get more complicated features working. Similarly, it's important to focus hard on the most important specifications and features before branching out into ones that are optional or make things look nicer.

Advice for Future Teams

We have two particular pieces of advice for future teams. Our first is to really spend a good amount of time at the very start of the project researching tools and technologies available that might be good to use in your project. We didn't do as much of this as we probably should have at the very beginning, and because of this needed to go back and change things once we found tools or technology we wanted to use. This ended up taking a fair amount of time, and we could have avoided it if we started with that technology from the beginning. Second is to ensure your functionality works at all before taking time to optimize it. The last thing you want is to discover your optimizations are actually the reason the functionality isn't working as expected, or at all. So just make sure things work the way they are supposed to, and then go back and try to make changes to optimize things.