{Christina}

Hi everyone! I’m Christina from GitGoing, and I’m joined today by Michael, Travis, and Sarah. To recap from last time, the goal of our project is to develop a web-based code review tool integrated with Git. When last we left off, GitGoing was in-progress with several functionality requirements still needed, such as automatic retrieval of diffs when a file is updated, implementation of an approve and reject system for reviews, and a display of file history for projects. We also needed a way to comment on multiple lines of code as well as a notification system when inviting users to projects. Well today, we’re happy to present that many of these planned features are now live, and available for use.

We’ve talked about it before, so we won’t go too in-depth, but here is a quick recap of where we were previously. We had our log-in and registration systems in place, which is fully authenticated by Congito from AWS. These calls are defined in our javascript code through the AWS service Amplify, which is in turn paired with Amazon Cognito. Through these services, we may include other related services, such as the simple email service, and our simple queue service. Additionally, our projects page displays file contents, specific user data, and code diffs after securely retrieving from a DB using Cognito credentials. Here is a quick look at our database design. We’ve also successfully implemented our inline commenting, where a user is able to comment on a specific line of code. Using JavaScript react, a comment is developed as a single component where each component has a separate comment box which contains the line of code, comment list, and username. Once a button is clicked, the comment box will be returned with the proper information needed to post a comment.

Since then, we’ve developed a lot of new features into our site, so here’s a look at our progress and where we are now. Diffs are now automatically retrieved when a file is updated. Using a flask app, we are able to retrieve the diffs between the current file and the proposed updated file when the user uploads a new file. Users are now also able to approve and reject these changes since we’ve stored the diffs as a review status in the MySQL database. This allows users to access review changes, and approve a review which updates the file table. A display of the file history for each project can also be seen. As you can see, you can view the file name, file type, and the date of the modified file. Multiline commenting is also implemented into the site where users can highlight multiple lines of text and insert a comment, which is then stored in the database. Lastly, we have our notification system up and running where users will receive a notification when they are invited to join a project.

In order to achieve these new features on the site, we did a lot of research and testing. For example, I personally did a lot of research into how react components work in order to develop the multiline commenting and I know my teammates also did a lot of research into how to store information into the database, how to develop a flask app, and how to redesign the look and feel of the overall website through CSS. Altogether, we worked hard developing and testing what we have to finally have our new features deployed on the site. Now I’m going to hand it off to Sarah to go further in depth about our progress, achievements and our bumps along the way.

{Sarah}

Thank you, Christina. Hi everyone, this is Sarah. So, you’ve just heard about our overall work which includes the implementation of all the features that make up our website. Although, some of our planned components are live and available for use, others were left off for the simple reason that they did not work as wished.

For instance, in our code reviewer project, one of the major requirements is to track and display the differences between changes made on a file. To achieve this, we needed to find a way to create a tool that could help us retrieve the diffs. So, based on simple trials and errors, we opted for using Flask which is a Python framework because of its simplicity and high scalability rather than the JGit Library in Java.

Another complication that we run into was the expansion of file formats for a review. Meaning that file reviews did not work for nonstandard file types such doc files, pdfs or even images.

One other unsuccessful element is that we couldn’t achieve consistency while splitting the diffs either into full context or minimal context and we just ended up leaving it off and replacing it with another feature that can render the diffs as either unified or split, by using a toggle switch.

Additionally, adapting our website to different types of screen sizes and resolutions is something that was achieved as well as the integration of our notification system into the live build which we kinda ran out of time on.

Okay. That being said, let’s switch the topic and talk about our success. Despite the complexity of the task, we managed to meet as many requirements as possible, such as the implementation of our authentication system, the database system, inline and multiline commenting, diffs retrieval and display, approval and rejection system for reviews as well as the notification system for the user invite. Although all these were successfully built, our biggest accomplishment remains the fact that we were able to host all those components on AWS. So, let me pass it to Michael and he can walk you through all the challenges that we faced on our road to success.

{Michael}

Thanks Sarah, and hello everyone, let’s talk about some of our biggest challenges of capstone development. Firstly, the biggest challenge, which I’d also say is one of our biggest accomplishments, is securely hosting our fullstack application, with a domain users can actually visit. I would say that this took up more research and development time than anticipated, and any features we were hoping to get implemented and didn’t were largely because of that. But even though it was our biggest challenge, we got there, and we’re happy we were able to pull it off. Tying into that, a lot of challenge was just learning new technologies. A lot of times you know exactly what you want to do, and the logic of your solution isn’t the confusing part, but how to set up that logic within various technologies is where things get tricky. So, I wanted to talk specifically about some of those things that we spent a lot of time learning about. Obviously, our frontend is a React frontend, so we had to spend a lot of time with this, more than anything else, probably. For me, this was also my first time coding in JavaScript at all. And despite spending so much time with React, I would say that this is still one the biggest unknowns to me. It just feels like there’s so much more to learn still. Another technology which is equally important, and used across all aspects of our application, is Amazon Web Services. Diving into AWS with no idea what you’re doing is truly terrifying, and I would not recommend it to anyone. Finding some tutorials or taking some certification exams before you’re trying to launch your app would make this part a lot easier, but after capstone and Alfred’s Cloud Computing class, I feel like I have a much better grasp on this, and we’ll touch on that again in a little bit when I talk about things we’d do differently. These next two things we spent much less time learning, but they were still new for most or all of us, first is APIs, which none of us had written any before, but after already being exposed to several software patterns and it being a fairly inutitive interface, this didn’t take much time. Lastly, again this isn’t a new technology by any means, but I don’t think any of us were particularly experienced with CSS, so it may as well have been an alien language to us. Another big challenge for us sometimes, I think, was just trying to understand too many moving pieces at once, which can be very overwhelming and really slow down progress when compared to breaking down problems into small, digestable pieces.

And launching off of that, I’d like to talk about some of the things that we would do differently next time around. First, a pretty large chunk of our time was lost to debugging and testing, and setting up continuous integration and deployment would have helped us reduce that by a pretty big margin, automating many test cases, so when we merge new changes we know we aren’t breaking features. Next, the deeper we got into frontend development, the more apparent it became that it would benefit greatly from some sort of state manager such as Redux. When dealing with multiple nested components, drilling state values all the way down yourself can be very tedious and inefficient development time spent. In a similar vein, we also would have used more libraries for UI elements, because creating our own is somewhere we lost a decent chunk of time. This is relatively minor, but we have two APIs when the node.js app could serve all of our needs. I mentioned this earlier, we would definitely use more AWS services. There are some things we’re doing such as some simple file storage in a database which would be better suited for something like an S3 bucket, or operations which could utilize Lambda functions rather than a virtual machine. In terms of coding practices, we definitely all could have followed better naming conventions sometimes, been more careful with merge conflicts, avoided overwriting changes, stuff like that, and I think we all could have better managed our time and energy, planned better, broken down problems better, but with all things considered, I’m really happy with how everything turned out, and really proud of everyone on our team. And with that, I’m going to pass it off to Travis to demo for y’all.

{Travis}