FINAL PRESENTATION SCRIPT

Person 1:{

//todo

}

Person 2:{

Thank you Sarah. My name is Christina and I’m going to cover our design architecture and what technologies and techniques were used to build our project. GitGoing is implemented as a Client-Server pattern where the server layer will be hidden and solely used to listen to ongoing requests from the clients on the backend. A Client-server architecture of a computer network allows many clients to request and receive service from a centralized server. Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. As shown on the screen, this is our design architecture for GitGoing.

In this case, the Client layer will be a Web browser on the frontend built through HTML, JavaScript, and React Frameworks. The output of these will essentially be what the client or user can see which is the actual login/registration system, notification system, and many other pages that we will be implementing. We will later see how these dependencies work and how they are used to build the projeact.

The client layer will obviously be connecting to the server layer through the internet. The Server layer will be hosted through AWS, and built using API gateways and Python/Flask frameworks. Both components works to respond to ongoing requests from the server side. Cognito, a user authentication tool within AWS will be used to enable authentication to requests. To process these requests, GitGoing uses Lambda and a MySQL database to do so. Altogether, two parties (client and server) work collectively to process, edit, store, and deliver an operable system to clients whether it be on a PC, smartphone, or laptop.

Here are a list of technologies, and techniques we used to develop our project. Each dependency is largely intertwined with the other to ensure a fully functioning system. As I mentioned before, we used react to implement most of the frontend components alongside javascript and html. React is a UI library created by Facebook that helps you create interactive web applications made up of components which is written as a plain JavaScript text.  Through react, we were able to create a successful login/registration system as well as a notification component.

The login/registration system is also paired with cognito, which is an Amazon tool that provides authentication, authorization, and user management for web and mobile apps. For example, users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, Google or Apple. However, for this particular project, cognito is specifically just going to be responsible for pushing new user information into the database and ensuring authorization.

Information and data from the user will ofcrouse be stored in a MYSQL database for reference. We spent a lot of time developing a way to add, retrieve, and delete such data by creating an API using node.js and express. Node.js is  an event-driven development platform for executing JavaScript code and Express is a flexible Node.js web application framework. Using Node.js and Express, we were able to create an API to communicate with with the database. The database and web application itself is all going to be hosted through AWS and we hope to continue to use these technologies to expand our project. Now, I’m going to hand it off to Michael to talk about some code that was developed through the use of these technology tools and what we’ve accomplished from them.

}

Person 3:{

//todo

}

Travis:

“Thank you Person 3(I think it’s Michael right?). Welcome everyone, I’m Travis and I’ll be walking you through a short demo momentarily.

As you can see here, I’m first presenting the three cmdprompts powering Git Going. A flask python REST api, a nodejs/express REST api, and a standard react APP. These power the website, as you can see here. This is our login page, and we’ll run some tests on it. First, the standard user input on every website, the empty case. It throws an error, as it should. Next we try without a password, and again, another error. Now we fill in both fields and get an incorrect username. In the code, the comparison of user input is against a literal string, but the logic is in place to make an authorization API call. The server I use here isn’t robust enough to handle it on its own, yet. Now with both fields correct, we pull up the localstorage to show how the site tracks login state. While this value is true, we’re loggedin, but if it’s deleted or false, it redirects us to the login page. As you can see, we can’t access certain pages without this state. Of course, users don’t want to delete something in local storage when they want to log out, so we give them a log out button! This sets their state to logged out, returning them again to the login screen. Now we’ll look at user registration and how it pushes entered data into our AWS hosted MYSQL database. Here you can see our latest entry, as well as many of my trial runs recording the demo, lol. Now we’ll also check for standard error messages for incorrect fields. It all checks out so we’ll click the redirect to home. This obviously should happen automatically, but again, the API I designed isn’t robust enough to handle the redirect itself yet. Now to show you guys the primary functionality of the site, a code review that displays the diff between two files! First we select our pullrequest.json file to structure the data as a dictionary for the API request. \*pause\* And if that sentence confused you, ask Michael about it after the fact, because just hearing it from him the first time made my brain whimper a little. \*unpause\* With all that taken care of, the code review is created on a file, previously defined, with some changes I’ve already made! As you can see, there are – symbols to indicate what’s been removed, and + symbols to indicate what’s been added. Now we’re make some changes to the same file ourselves!. As you can see, the new diff is generated on our site! Now let’s take a look at the flask API that’s been running the entire time, listening for requests. As you can see, I’ve made a few diffs already. It’s pretty neat, huh? Now to wrap up, let me demo for you, just a little, how the express api can handle DB requests through the browser! First I pull everyone in users. Quite a list, we saw it before. Then I check for admin, which maybe wasn’t the best idea because there’s no one there. And finally, I do a check for a name I’d been using in previous test cases. And it all checks out! That concludes the demo, and

now I’ll take a moment to talk about the road ahead. Obviously, we have a lot of work ahead of us still, but I am proud to be here at the end of our first Capstone quarter with my group. We’ve all grown so much as students, as developers, and as friends. And as we take a moment to recharge, and reflect, I know we’ve already got our eyes on what’s to com. As demonstrated, certain key features still require implementation; Integrating Christina’s amazing notification system into our review system. Generating inline commenting and messaging on reviews. Having a login system that actually, y’know, authenticates people. Our database API needs to be much more robust, capable of securely retrieving information. This and more lie before us, but I gotta say, look at how far we’ve come.

Thank you, very much everyone. I’ll now open the floor for questions.