FINAL PRESENTATION SCRIPT

Person 1:{

Hi everyone, my name is Sarah, and I am joined with my teammates Christina, Michael, and Travis. Today we will be presenting our senior capstone project which is the Bellevue College Code Reviewer.

For the achievement of this presentation, we referred to the documents we created during the quarter such as: our Press Release/FAQs, the Software Requirements Specification (SRS), the Software Design Document (SDD) and our Testing Log.

Without further ado, let’s quickly jump to the actual project by defining the code review:

What is code review?

**Code Review** is nothing but the systematic examination of the source code, generally used to find out the bugs at early stages of the software development. Code review is important for the fact that it helps limit the risk of unnoticed mistakes, it dramatically improves the code quality by making it more efficient, it also makes everyone better with the feedback received from others, and finally helps developers familiarize themselves with code they haven’t written but might be asked to maintain in the future.

This entire process is very critical and therefore requires the utilization of appropriate tools. Those tools make it easier in many ways, but many of them are usually either internal to companies and limited to whatever platforms those companies support or they require some extensive setup.

This is the reason for us to come up with our code review software, named Git Going. You may be intrigued by this name, especially the word Git, but the simple reason for that is because our software is integrated with Git which is a distributed version control system that most of us are familiar with.

The actual goal of Git Going is to make it easier for developers to collaborate and peer review codes from any device and any location. And we can achieve that in a few ways.

First, we provide a web-based platform with a secure two-factor authentication, a registration and user login system which can be accessed from a multitude of devices.

Our web application also has an account setting as well as a file change system which allows users to upload new files or replace existing files, where replacing an existing file will require one or more approvals from others. And when an update to a file is proposed, we give reviewers a compact and intuitive display of changes from which they can leave comments, submit an inquiry, approve, or decline changes.

Additionally, within the platform we provide the user with several features which help in facilitating collaboration and peer review. One of these features is the ability to organize files as projects, which allows project owners to invite other users and assign them tasks or request review from them.

Ultimately, we provide the user with a friends list in which they can message each other and receive notifications for important project updates.

Those were the introduction, the description and the user functional requirement of our project, and at this point, I will hand over to Christina who will take you through the architectural design as well as the essential techniques and technologies used in our project.

}

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 project.

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 work to listen to client requests on the server-side. Cognito, a user authentication tool within AWS will be used to enable authentication to requests. To process these requests, GitGoing is planning to use Lambda and a MySQL database to do so, however we are currently using a narrow express API at the moment. 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 is 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 are 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,a 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. While we’ve created a working instance of Cognito paired to a generic react app, it is not yet implemented on our site.

Information and data from the user will of crouse 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 the database. The database and web application itself are 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:{

Thanks Christina, and hello everyone. I want to take you through what we think are some of our biggest achievements for this quarter. First, we have a partially functioning login system written from scratch, which successfully authenticates users, tracks and remembers login status, and prevents users who aren’t logged in from accessing pages they shouldn’t be able to, but is still missing a couple features such as password resetting. Travis will demo all this for you soon.

At the same time, we figured with people being assigned to reviews, or having their reviews approved or rejected, it was obviously important to have some sort of way to notify users of those events, so Christina built a notification system we can hook to events we deem important.

In parallel to all this, of course we have a fully functional registration system, because how can you sign into an account without it existing? This asks the user for a unique email, username and password, it verifies the two passwords match, and then pushes this information into an AWS hosted MySQL database using a POST request implemented with nodejs and Express.

Being code review, we also obviously have to show the users some kind of difference between file versions, so we’ve also set up a Flask app which receives file and directory information from the user through a POST request, and then uses that information to return the user a diff from a remote location.

Now, of course along with accomplishments are failures and frustrations. While we don’t have any automated testing yet, we plan to in the future, and in the meantime I wanted to share some of our biggest headaches that took us the most time to work through.

Firstly, conditional rendering. Because of how much time Travis spent on the login and registration systems, he would be better to ask if you have further questions, but simply put, it’s just the idea of having boolean conditions tied to the display or functioning of various user interface components.

Another headache was excessive (and also unsupported) dependencies, because it turned out in a few cases, especially early on figuring out how we were going to deal with our diffs, we had to test library after library only just to end up saying “Ok, we’ll do this ourselves.”

APIs. Actually not that bad once you get used to it, but none of us had written an API before this, so learning it in parallel with everything else was still somewhat tricky at times. For example with my Flask API, a simple one extra line in package.json to redirect requests to the correct port, which took some time to figure out.

Lastly, and I’m sure everyone here can relate to this, unexpected problems. I’m talking the kind you never even would have predicted. One example I have is when I tried converting my current UI elements to React Native, which took a few hours to figure out on its own, only to learn the only file picker I could find only works in development, and not in the Windows debug environment. So I frustratedly threw my hands up and put it on the back burner for now and ended up right back where I started. I wanted to share this because I’m sure you’ve all had experiences like this, you’re not alone. But with that, I’m going to pass it off to Travis to present it all for you.

}

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.