CSE 216 Project Manager Report: Phase 3, Team Liger

List your team members, providing name, email, and role for this project. Also provide the URL where your live Heroku-hosted web front-end can be found.

Trent Gray – tcg219 – Project Manager

Jack McBryan – jnm219 – Android

Mira Straathof – mts219 – Admin

Joon Seonwoo – jos219 – Backend

Conrad Ozarek – coo219 – Web

URL:

https://quiet-taiga-79213.herokuapp.com/

Back-End Server

Describe the changes to the set of REST API endpoints that your team agreed to implement.

The main changes that were made to the set of REST API endpoints that our team agreed on were that a post route was added for checkLogin, a get and a post were added for comments, a get and a post were added for register, a post was added for login, a post was added for both upVote and downVote separately rather than one route for both functions, a get and a put were added for profile, a put was added for changePassword, and a post was added for logout.

Did your data model deviate significantly from the model discussed in class? If so, why?

For the most part, our data model closely stuck to the one discussed in class. The only changes made were that we had two separate tables for users, one for registered users and one for unregistered users to aid in the verification process, the addition of a separate table just to hold user profile information, and the use of username as a

unique foreign key instead of a user id. We also added checks to make sure username is always unique to ensure it is in fact a unique foreign key.

Were there any issues that kept the back end from reaching 100% functionality? If so, what were they?

There were no issues that kept the back end from reaching 100% functionality. We were able to have the backend match and fulfill all of the functionality required for this phase of the project.

Is the back-end code appropriately organized into files / classes / packages?

Yes, the backend is properly organized. It has separate java spark calls for different functionality, separate request and response types tailored to individual tasks. The code is effectively split up and organized so that it is easy to see and understand how each call and bit of functionality works.

Are variables, functions, classes and packages named well?

All classes, variables, functions, and packages were very well named in that they were not excessively long while being long enough to give enough insight and information as to the nature and function of the variable, function, class, etc. Many of the variables and class names are cohesive and consistent between branches since we thoroughly discussed names as well as created a google doc which contains all of the chosen and accepted names for the project.

Are the dependencies in the pom.xml file appropriate? Were there any unexpected dependencies added to the program?

Throughout the course of this phase, no changes were made to the dependencies in the pom.xml file and all current dependencies in it are properly used. Therefore, all dependencies in the pom.xml file are appropriate.

What was the biggest issue that came up in code review of the back-end server?

While it is rather small, the biggest issue that came up in the process was fixing the createTime variable. While looking at the code and its result, we found that the time was both in military time and set to a different time zone than eastern standard time. After some work, we managed to edit the time and have it reflect a more accurate time.

What technical debt do you see in the current back-end server implementation?

As far as I can tell, there appears to be no obvious technical debt in the current back-end server implementation. All of the code appears to be structured well and designed with the idea and capability to handle further expansion in the future.

Describe any refactoring that was performed to reduce technical debt from the last phase

The only refactoring that was performed was the changing of how upvote and downvote worked from the last phase. Instead of using one rest route to handle both upvote and downvote, two rest routes were made, one for each. We also changed the number of votes from an integer in the message table to being two separate tables, one for upvotes and one for downvotes, that kept a record of who upvoted and downvoted each and every post.

Are the unit tests for the back-end server complete/satisfactory? If yes, why do you have confidence in your answer. If not, what's missing?

After looking over the backend tests, I believe the back-end server has satisfactory tests. I have confidence that this is the case because, while looking at the test, I can see that a test exists for every functionality of backend and appears to thoroughly test each one.

Web Front-End

Did you use MVC to organize your web front end? If so, did it help? If not, why not?

Yes, MVC was used to separate the visual aspects from the control aspects in the front end. Model was also split up, but it was handled by the backend. Using MVC was helpful overall because it helped organize the code as well as helped separate tasks. It also helped simplify implementation, reducing "spaghetti" code.

Does the user interface match the interface for the Android app? Why or why not?

Yes, the user interface does, for the most part, match the interface for the Android app. The only notable differences between android and web is that web has a profile page, which is specified in the instructions for this phase for web to have and not android, and that android has a conformation page for logging out. When hitting logout, android has a check and asks if the user is sure. The user can then hit confirm and log out or cancel and return to the app. In web, hitting logout automatically logs the user out without any conformation afterwards.

Is the web front-end appropriately organized into files / classes / packages?

Yes, the web front-end is appropriately organized. It has different files for different visible objects as well as for different states, each with their own classes and methods. Overall, the organization is quite good and is easy to understand when looking through it.

Are variables, functions, classes and packages named well?

Yes, each variable, function, class, and packages is given an appropriate name that is easy to read as well as to understand. Many of the variable names match the ones we as a group chose and put up on a google doc to record as stated before.

Are the dependencies in the package.json file appropriate? Were there any unexpected dependencies added to the program?

No new dependencies were added in phase 3 to package.json and all dependencies in it are used. Therefore, all of the dependencies may be considered appropriate.

What was the biggest issue that came up in code review of the web front-end?

After working through the code, the biggest issue was that, although it was working before, comments were not working in the web front end.

What technical debt do you see in the current web front-end implementation?

After looking through the design of web, there appears to be little to know technical debt as far as we can tell. Everything appears to be well coded and working properly.

Describe any refactoring that was performed to reduce technical debt from the last phase

As there seemed to be no technical debt found in the code from the previous phase, no technical debt appears to have been refactored. Everything from the previous phase seems to be working well and was able to be successfully built off of.

Are the unit tests for the web front-end complete/satisfactory? If yes, why do you have confidence in your answer. If not, what's missing?

The tests for the web front-end are complete and satisfactory. I have confidence in this answer because, after looking through the tests, each method and functionality is called and tested, with easy and convenient naming of tests to easily differentiate between all of them, and, from as far as I can tell, every method is rigorously tested.

Each method is tested to see that it works when it is supposed to as well as tested in ways that should throw error while checking for these errors.

Android App

Describe any significant deviations from the UI design discussed in class.

Overall, there weren't any significant deviations from the UI design discussed in class for the android app. The only deviations made were that comments is a button on messages which takes the user to a page that is filled with all of that messages comments and that there is a logout confirmation page. Other than these minor deviations, the android app replicates the UI design discussed in class.

Is the android app appropriately organized into files / classes / packages?

Yes, the android app is appropriately organized into files/classes/packages. It breaks up tasks and special functionality into separate files and methods in a way that makes things easy to comprehend and doesn't overcomplicate the entire process.

Are variables, functions, classes and packages named well?

All of the variables, functions, classes, and packages have been given extra time to be named specifically to make understanding how android works as simple as possible. All names effectively aid the reader in comprehending the purpose and functions of every variable, function, class, and package within the code.

Are the dependencies in the build.gradle file appropriate? Were there any unexpected dependencies added to the program?

No new dependencies were added in the build.gradle file during phase3 and all dependencies within the build.gradle file are used appropriately in the android code. Therefore, it is safe to deem all of the dependencies in the build.gradle file appropriate.

What was the biggest issue that came up in code review of the Android app?

The biggest issue that came up in the code review of the Android app is that, while there was nothing wrong with the actual code itself, there was an issue with pushing the retrieving the code on a different computer using git. While the original copy of the android app worked fine, all copies of it retrieved from git had an odd variable error. After some debugging, we did manage to get everything working and effectively pushed everything to bitbucket.

What technical debt do you see in the current Android app implementation?

As far as I can tell, there appears to be no technical debt in the android app implementation. Everything seems to be coded in an efficient and scalable manor.

Describe any refactoring that was performed to reduce technical debt from the last phase

While it may not entirely be considered technical debt, a lot of the old variable names from the previous phase had to be changed. In light of all of the new features that were added to the social media platform, such as comments, users, likes, dislikes, etc., a lot of the old variable names simply didn't help discern or properly describe many of the variables and classes they named. As a result, many of the old names had to be changed in order to help them differentiate from other variable and function names.

Are the unit tests for the Android app complete/satisfactory? If yes, why do you have confidence in your answer. If not, what's missing?

The unit tests are complete/satisfactory. All of the tests in android test every bit of functionality that the android app contains thoroughly. As a result, I have confidence that the unit tests are complete and satisfactory. Many of the tests were also broken up by what functionality they test, making it an easy task to see what features are and aren't tested and, as far as I can tell, all of the functionality appears to be properly tested.

The Admin App

Describe the process that your team decided upon for managing the process of registering and activating users.

For this process, we decided, for the verification functionality, to add two user tables. One containing authorized users and the other containing unauthorized users. In registration, users give a real name, username, and an email. They are then put into the unauthorized user table. Once authorized by admin, an email is sent to the user's email, giving them their username, which they chose, and their auto-generated password. Once they login, a default program is made for them. Also, the user can, at any time after their initial login, choose to change their password instead of having to use their autogenerated password all the time.

Is the Admin app appropriately organized into files / classes / packages?

All files, classes, and packages are appropriately organized. There is one file for the main functionality of the admin app, one for handling database changes and functionality, one for making an auto-generated password, and one for sending emails. As a result, everything is neatly separated and organized.

Are variables, functions, classes and packages named well?

All of the variables, functions, classes, and packages have very self explanatory and easy to understand names to them. As a result, I am quite happy with them and find them to be well named.

Are the dependencies in the pom.xml file appropriate? Were there any unexpected dependencies added to the program?

In admin, a dependency was indeed added to the pom.xml file. The dependency added allowed admin to use sendgrid to be able to send emails to users during the authorization process. Apart from this dependency, no other dependencies were added or removed. Therefore, the dependencies in admin can be considered appropriate.

What was the biggest issue that came up in code review of the Admin app?

The biggest issue that came up in the code review of the admin app actually had little to do with the admin app itself. The issue wasn't actually caught in the code review of the admin app itself, but instead in the code review of the backend. While going through the backend tests, it was discovered it would never work because, during the admin app tests, the admin app never left the test server in a condition where the backend could then test. After finishing its tests, the admin app would drop all tables, leaving nothing for the backend to work and test with. After discovering this issue, the admin app was changed so that its tests left the test server in a state the backend could then work and test with.

What technical debt do you see in the current Admin app implementation?

After looking at the admin code, the only possible source of technical debt that could possibly be found was in the random password generation process. In the password generation process, the generator picks a random Napoleon Dynamite character name, or the name Spear, and adds three random numbers to the end of it. While amusing, this process may not be considered the most professional or the most effective. However, I personally asked for this to be coded, as I thought it would be amusing and add some personality to our code. As a result, I will take full responsibility for this design decision.

Describe any refactoring that was performed to reduce technical debt from the last phase

Since the code from the previous phase worked quite well, there really wasn't any refactoring needed in this phase. We were easily able to build off of what was previously coded with no trouble whatsoever.

Are the unit tests for the Admin app complete/satisfactory? If yes, why do you have confidence in your answer. If not, what's missing?

All of the unit tests for the Admin app appear to be complete and satisfactory. In addition to checking through them once, I check them again after they needed to be changed to allow backend to do its tests. As a result, I am fairly confident that the tests work after looking through them several times.

Project-Wide

How well did your team estimate effort?

Overall, I think my team did pretty well with estimating effort. We weren't perfect, but we did expect it to be a lot of work, which it was. As a result, we were working hard as a group as much as we could together in order to make as much headway as we could every step along the way. We still ended up with quite a bit of work to do at the end, but it wasn't more then we could handle.

What techniques (team programming, timelines, group design exercises) did you use to mitigate risk?

For the most part, our group used team programming and a few group design exercises. Throughout the phase, we met as a group as much as possible and coded together as much as possible to work through problems and collaborate as well as be able to test each others code along the way. We also used group design exercises such as writing tables on a white board, having group discussions on design choices, and having a google doc containing common variable and method names.

Were there any team issues that arose?

For the most part, there weren't really many team issues that arose. The only problems that occurred were when someone needed to have another group member do something with their code so they could run and test theirs, but

were unable to get ahold of that group member. However, since we all met together so often, this issue was a rare one. Plus, we were always on GroupMe, so when we weren't together, group member response time was pretty quick, even at odd hours.

Describe the most significant obstacle or difficulty your team faced.

The most significant obstacle we faced was getting the whole login, logout, and registration process working. Since so many tasks were so intertwined between all of the different parts of the project, getting everyone to all work together correctly was a herculean task. Especially when it came to debugging. Since so many different branches were working together on the same, single task, finding exactly which branch was causing the error was sometimes a game of hide and seek.

What is your biggest concern as you think ahead to the next phase of the project?

My biggest concern is that we may have to alter our coding practices. Up until now, our group has depended on getting and working together simultaneously. In the future, however, I know that different people won't be around at different times, meaning that we won't all be able to work together all the time. As a result, much more effort will have to go into scheduling and working out a game plan on who should work on and accomplish what by when in order to allow everyone to make progress and to prevent any one group member form holding the rest of the group members back from making progress. I also fear unforsean technical debt which will reign its ugly head in the future, though I am not yet sure if this is a fear spurned on by little more than paranoia, or a substantial, and accurate worry. I guess we'll just have to wait until the next phase to find out.