Features and Cuts

We completed almost all features that were listed in our original SRS. One feature we decided to cut was showing a history of content a user has submitted. We have the back-end implemented for this, but did not build the front-end in order to focus on making the existing features stronger and more user friendly. The number of users who will submit content compared to those who will use it for rating issues and viewing candidate similarities is very low, so it was low on our priority list. We estimate the time saved from this cut to be between one and two developer days, one to build the feature, and possibly another if bugs were discovered.

Initially we had intended to support both Android and iOS. However, it soon became apparent that our team did not have the resources required to consistently support the iOS platform. Many members of the team both did not have iPhones, nor the iOS development environments of the closed Apple ecosystem. We therefore had to rely on a very limited set of people to build and support the operating system. Android, by contrast, was accessible and testable by all via open source emulators and development environments. We estimate the iOS cut to have saved us several developer days.

We did not implement the stretch features we outlined in the SRS. They are all very large features that are high risk because they would most likely introduce more bugs and could set us backwards. Instead, we focused our efforts on a couple smaller features that were not part of the original SRS.

A feature we added was a front-end UI feature for crowdsourcing content approval, something we had initially intended to be a command line utility. However, upon further exploration of this topic, it became clear to us that in order to become a well respected platform, and reach the amount of information that we hoped would be critical to provide the user with new information about politics, the amount of crowd sourced content would have to be staggering. Therefore, in order to approve that much content (since approval is necessary to order to keep a high quality level), we needed to rely at some point on crowd sourced approvals. Therefore, we decided to develop and implement this feature in our releases. This was done both in order to test its feasibility - as a proof of concept - in addition to allowing us to test this feature during our development stages. As opposed to adding it to the application at some later point, and therefore providing us with less opportunities to use it and critically think about the user interface decisions involved with its creation.

Another feature we added was showing how similar a candidate was to the user, instead of showing just a ranking. This feature was requested by our customer after the beta release.

Our initial schedule was extremely aggressive, and we quickly realized that the tasks for each week were unrealistic. The front-end team relied heavily on the back-end team, and after the back-end was completed, the front-end would realize several changes that had to be made in the back-end. There was a lot more back and forth between the front-end and back-end teams than we had originally anticipated, which didn't leave time for members to work on stretch features.

We spent a lot of time setting up the webdriver tests to automate testing for the front-end, which

didn't gain us very much because our front-end kept going through iterations and all of us were dogfooding the application which surfaced most bugs. The time would have been much better spent working on additional front-end features. We spent too little time gathering legitimate content for our application. We used dummy data initially, but this became a burden later on when trying to clean the database when we were user testing and for our final release.

Task Assignments

Our team's current roles differed slightly from what we initially documented. The original team structure was front-end, back-end, and full stack, but the full stack team mainly worked on front-end features and became the crowdsourcing team. Aaron ended up working on architecture and testing frameworks instead of front-end components. This is what occupied the majority of each member's time:

Large-Scale Architecture / Testing Frameworks

Aaron spent a majority of his time on larger scale architecture changes in the codebase rather than detailed implementation in a particular team. Early on he set up the UI libraries and integrated them with React, and set up a build process for Phonegap. In the mid weeks his focus was on the Facebook authentication system and integrating it with Facebook. Toward the end of the quarter his time was mostly spent fulfilling the testing requirements of the class. He implemented unit testing, a continuous testing bot to automatically build and email results to the team, and the webdriver testing framework.

Front-end

Geoffrey spent the majority of his time implementing and applying bug-fixes to the Rate Viewpoints and Your Candidates components. This actually required significant collaboration with the back-end team, especially when unexpected bugs came up. The majority of Geoffrey's time was dedicated to fixing bugs, more in collaboration with the back-end team but also a significant number of bugs in the front-end. This deviated from his original expectation that he would be spending more time implementing React components. Aaron spent some time assisting the front-end team with React nuances, bug fixes, and some integration issues with the back-end. Roee spent his time working on the Political Profile component.

Back-end

The back-end team consisting of Nick, Riley and Todd worked as a group in the beginning of the project to learn firebase and create the initial slated classes and methods slated in the SDS. After the scaffolding was fixed, they met with the front-end team and realized that much more functionality was needed from the back-end team. That along with bug fixes for the front-end team took up most of the rest of the project time, along with streamlining the algorithm for candidate rankings.

Crowdsourcing

Sonja spent most of her time implementing the crowdsourcing submission form and approval utility. Ryan worked on getting the information submitted through the crowdsourcing feature to write to our database, and formatted / wrote all of our documents in Latex. Ryan also added features to the crowdsourcing feature that made submitting content a more enjoyable user experience (form validation and auto-submission).