SRS POSTMORTEM

FEATURES AND CUTS:

We ended up changing our delivered features quite a bit compared to our original plans. While the basic functionality of the app - the ability to trade items with nearby people - remains the same, many features were cut, with some being added in their place.

Our original SRS specified several major features for our app. The first was user profiles tied to Google Accounts. We have successfully implemented this feature in our app, as described in our SRS. We use Google's own sign in mechanism to sign the user into our app. Each user also has an inventory with tags and items, just as the SRS describes.

The next feature the SRS mentions is automatic matching based on tags and item preferences. This feature, along with searching items by tag, unfortunately had to be cut. It would have taken far too long to implement relative to how much time we had. In its place, we implemented a browse feature which would allow users to view other nearby users to trade with. It shows every user within 20 miles of the current user's ZIP code in their profile settings. This feature was much faster to implement than the original search features would have been, allowing us to deliver the basic functionality of the app in the limited time we had. Viewing other user profiles works like in the SRS. The user can click on the users in the results to go to their profile, where they can view the other user's inventory and preferred tags, as well as propose a trade. Proposing a trade works as described in the SRS as well, with users selecting items from both inventories and clicking propose trade in order to send the trade.

The last of our major features was managing trade requests. Managing trades works like the SRS describes, with incoming and outgoing trades being shown, as well as accepted and rejected trades. Contact information, however, does not work like SRS describes. Rather than being automatically swapped, the contact information of a trading partner is found by going to a completed trade and clicking a "send email" button. This was much easier and faster to implement than automatic emailing would have been. Considering our limited time in our last several weeks of this project, it was much more practical to make contact swapping manual than automatic.

Overall, we probably saved around 2 weeks by cutting searching and around a week by cutting automatic emailing (judging from how long other, much simpler features took). Considering our limited time, these time saves were crucial to getting our app working on time.

TASKS AND ASSIGNMENTS:

The final team assignments essentially stayed the same by the end of the quarter, Artem, Jared, and Kieran stayed on the front end, while Taylor, Danial, and Courtney stayed on the back end. Artem and Jared were mainly responsible for the UI and the android services, while Kieran focused on testing. Taylor was PM, and mainly focused on the back end architecture and API. Courtney and Danial both focused on implementing handlers, while Courtney dealt with unit tests and Amazon AWS for server hosting, and Danial dealt with Amazon S3 and AWS for database/image hosting.

For the front end, a lot of the time was spent figuring out the tools, trying to debug external libraries, and debugging the app itself. There were considerable difficulties getting the simulated tests to run as well, and in the end, couldn't get to run. This went against our expectations, as we assumed a lot of these tools would be intuitive and easy to setup, given their popularity, but instead were massive, complex tools that required significant time to understand properly. The simulated tests are a good example of spending too much time, as it took away many hours of work from the application itself, and in the end, weren't even functioning, resulting in a net negative productivity. That time should've gone to polishing the applications interface and UX, which didn't get enough time because of the time lost elsewhere.

For the back end, most of the time was spent implementing the API and all the handlers, but there was a considerable amount of time spent trying to make the variety of services to work. Amazon AWS and S3 proved to be very confusing in documentation, leading to lots of time spent reading and watching videos that ended up accomplishing nothing that we wanted. There was also some miscommunication issues when defining the API for the front and back end that took up more time later in development, because we had to redefine it. I'd say the API confusion was an avoidable mistake that we spent too much time on -- roughly an extra day -- which took away time from important areas such as testing. And detailing the test framework is probably the area that we could've spent more time on, as not every API call case was handled.