Process Description

Software Toolset

Our end goal is to be able to make a trading application for the Android platform. As such, we are interested in using the following tools in order to implement our idea:

Languages:

- Java- As we are developing an Android application, it is only natural that a large portion of our project will require the usage of Java. We will use the Android Studio IDE to develop the application and create the front end of the application. We also plan to use Android Bootstrap to aid us in our front end development, rather than create everything from scratch. We also plan to use Retrofit, a tool specifically built for Android and Java that allows us to treat HTTP APIs as Java interfaces, which will hopefully aid in writing networking code and connecting our front end to our backend.
- Python- We plan on using python to manage the back end, as its simplicity will hopefully keep our backend code clean and easier to manage. We plan to use it to interact with our APIs and database servers. We also plan to use the Flask microframework to aid with networking tasks. The reason we chose Flask over other web frameworks such as Django is because it is seen as more beginner friendly, and minimizes boilerplate code while allowing for convenient creation of easily configurable server instances.

Version control/bug tracking:

- Git- Git is the most convenient version control and bug tracking tool to use in this case because it's universally known, easy to use, and also has a lot of great secondary features such as support for continuous integration that will make the testing process run much smoother.

Database:

- Amazon Web Services + MySQL- AWS is a tried and true option for data management, and will also be useful in API server hosting. AWS has enough horsepower to handle whatever we need it to do when implementing our project, and is reliable as well. We will utilize MySQL because it's well documented, simple, reliable, and free.

APIs:

- Google Identity- We need to have a username and password system in order to implement our desired features, so rather than create one from the ground up, we feel it will be easier and far more secure to use an established system instead. We can manage identity and authentication by using Google's account API.
- Google Geocoder - In order to make sure that people are able to make trades with people physically close to their location, we need to utilize Google's Geocoder API to convert between ZIP codes and physical locations.

Group Dynamics:

Our initial project manager will be Taylor Yoon. We plan on organizing our group into a front end team and a back end team. The front end team will focus on working on the Android application itself, as well as aspects of design. The back end team will work on the API, writing algorithms for the matching system, and managing the database that the application will use. Both teams will need to maintain good communication in order to minimize issues when trying to fit the front end and back end together. We hope to allow people to switch teams in the early phases of the project if specific tasks arise that they have experience with, but ideally later on in the project team members will have found their niche. We want to avoid team members getting stuck in roles they don't feel apt in. The team dynamic of back end and front end should stay similar throughout the project.

In case of disagreements, we hope to employ a democratic process where team members take a vote after issues have been presented. We will accept the solution that gets a majority of the votes. If disagreements continue, we may also choose to consult the client for clarification. This ensures that the largest number of group members are happy, but also encourages flexibility and quick resolution of issues.

Schedule & Timeline

Week 1: Oct 17 -> Oct 21

The main focus was getting the Design Specification done. We also familiarized ourselves with the tools we would need to use later on in the course.

- Front End:
    - Familiarize with Java, Android Studio, and Android Bootstrap (Kieran, Artem, Jared - 1 day)
- Back End:
    - Familiarize with Python, Flask (Courtney, Taylor, Danial - 1 day)
- Entire Team:
    - Complete Design Specification (Taylor, Danial, Courtney, Artem, Jared, Kieran - 3 days)

Week 2: Oct 24 -> Oct 28

The primary focus was getting all the starting code in place. Frontend handled the android code for zero release, while backend handled the server code. We also set up CI.

- Front End:
    - Create a runnable startup page for Zero-Feature Release (Artem - 1 day)
    - Create product website (Kieran - 1 day)
- Back End:
    - Set up AWS (Courtney - 2 days)
    - Set up MySQL database (Danial - 2 days)
    - Implement basic API functions for remote servers (Taylor - 1 day)
    - Initialize CI test suites (Taylor, Jared - 2 days)

- Entire Team:
    - Design API for remote servers (Courtney, Kieran - 1 day)

Week 3: Oct 31 -> Nov 04

This week we focused on setting up aspects of the beta release still not in place for the feature complete release. We tried to set up the structure of the API and implemented some initial front end functionality.

- Front End:
    - Create login screen (Artem - 1 day)
    - Create profile page, with inventory and wishlist (Artem, Jared, Kieran - 3 days)
- Back End:
    - Create API outline and mock to dictate the API structure (Taylor - 2 days)

Week 4: Nov 07 -> Nov 11

This week, we focused on getting functionality related to keeping track of users and inventories in place.

- Front End:
    - Improve profile page, with inventory and wishlist (Artem, Jared, Kieran - 2 days)
    - Connect application to API Server (Artem, Jared - 1 day)
    - Create tests with Espresso Framework (Kieran - 1 day)
- Back End:
    - Create User Handler (Danial - 2 days)
    - Set up API and database servers to accept and store profile info (Danial, Courtney, Taylor - 3 days)
    - Create Item Handler (Courtney - 2 days)
    - Create Image Handler (Danial, Taylor - 1 day)
    - Create Trade Handler mock test (Danial - 1 day)

Week 5: Nov 14 -> Nov 18

Because this was Feature Complete Release week, we focused on completing the trade algorithm and search features on both frontend and backend

- Front End:
    - Create tag searching window (Artem, Jared - 2 days)
    - Create tests with Espresso Framework (Kieran - 1 day)
    - Create trade window (Artem - 1 day)
    - Create requests window (Jared - 1 day)
- Back End:
    - Implement search functionality (Taylor - 2 days)
    - Implement trade and request management (Courtney, Danial - 3 days)
    - Create backend tests (Courtney - 2 days)
    - Create deletion method in Image Handler (Danial - 1 day)

Week 6: Nov 21 -> Nov 25
Not much was done this week, as Thanksgiving break made it hard to coordinate tasks. Some commenting was done on the backend code.

- Entire Team:
    - Backend comments (Taylor - 1 day)

Week 7: Nov 28 -> Dec 02
To prepare for the Release Candidate, we tried to focus on finishing usability testing in order to shed light on some issues raised by our application, and tried to improve our existing code base through the code review.

- Front End:
    - Fixed several minor bugs (Jared, Artem, Kieran - 1 day)
    - Improved look of UI (Courtney - 2 days)
- Back End:
    - Cleaned up backend code for code review (Danial - 2 days)
- Entire Team:
    - User Testing (Courtney, Taylor - 1 day)

Week 8: Dec 05 -> Dec 09
Since the Final Release is Dec 06, there won't be much work towards the application itself.

- Entire Team:
    - Prepare for demo (Artem, Jared, Courtney - 2 days)
    - Create postmortem SRS schedule (Everyone - 2 days)

Cut Features / Unreached Stretch Goal Estimates:

- Auto matching trading (Estimated to take at least 3-6 extra days)
- Searching by item (Estimated to take at least 3-6 extra days)

Risk Summary

There are multiple risks regarding the completion of the project. The first risk would be having difficulty setting up the Amazon Web Services. This is necessary for storing information such as profile inventory data. However, given that Amazon Web Services is a popular, reliable service, finding help for troubleshooting should be relatively easy. An example of someone who could help would be Amazon's technical support.

A second risk would be a miscommunication between the frontend and backend teams. For example, one team might not understand the objectives of the other team. This would cause confusion, which would most likely lead to a non-functioning mobile application. We can

minimize this risk by abiding by the documentation that the team has written for each part of the project. Consistently attending weekly meetings and communicating through Slack chat will minimize confusion and decrease the chances of miscommunication.

The third risk would be running out of time and not being able to add all major features. Our current schedule shows that it is possible to get all major features completed if we meet our deadlines. However, we are ready to cut features if time is running short. The major feature that we would cut first would be the automatic matching, simply because this is a convenience to the user. Cutting this major feature would not get in the way of the user achieving their goal, which is to trade items with other users.

Another possible risk would be that a team member does not complete their objectives in time. Communication between team members and notifying the team as soon as possible is the best way to avoid this. By having good communication between team members, this will allow team members to assist other team members' with their objectives, making it less likely that the team will have to cut features.

We a have a group of HCDE students ready to give an external user evaluation. An external user evaluation would be useful at multiple points. The first would be the Zero feature release for feedback on the user interface. The second would be the Beta release for feedback on the features and functionality. The last would be the feature complete release for bug testing.

Design Changes and Rationale

While building our application, we found that the Google Location API was not what we needed to keep track of user locations. Instead, we needed the Google Geocoder API. Using it, the user could enter their ZIP code and we could immediately convert it to geographic coordinates. The Google Location API could only give us the user's current coordinates, meaning the user would have to be at home in order to set their location settings. The Geocoder would allow them to set it anywhere.