How to Use this Template

1. Create a new document, and copy and paste the text from this template into your new document [Select All → Copy → Paste into new document]
2. Name your document file: "**Capstone_Stage1**"
3. Replace the text in green

GitHub Username: crowduckseq4

GarageSale

Description

Some people's trash is other people's treasure! Make money selling your junk and create more space at the same time!

Summary

This app connects owner that wants to get rid of their junk to a buyer that is interested with the treasure. Make money and clear up space, killing two bird with one stone!

What does it solve?
Helps people sell their old item(s) for cash and opens up more space.

Intended User

This app is for hoarders, people that would like to sell their old collections, people who are looking for used items.

Features

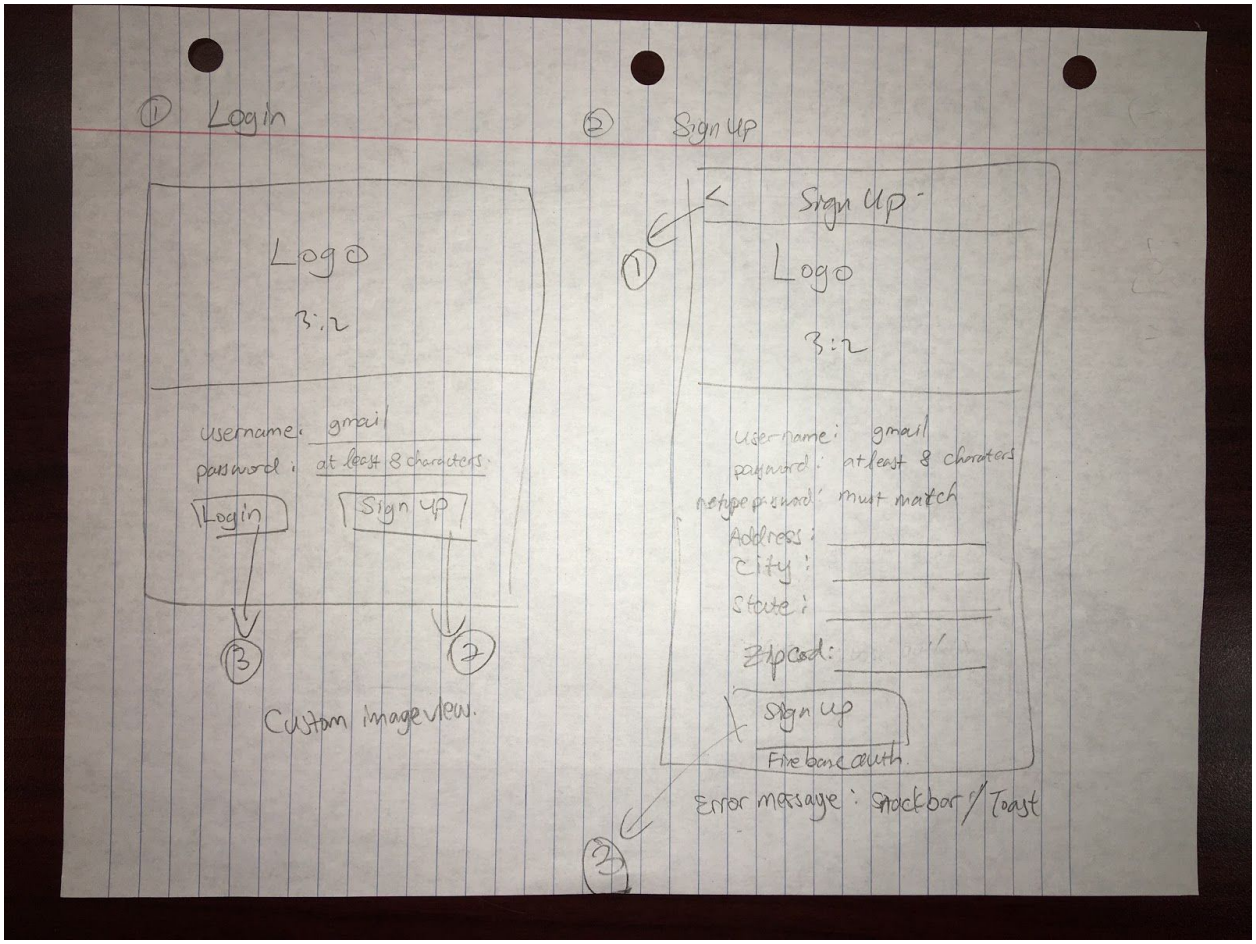
List the main features of your app. For example:

- Saves information
- Takes pictures
- Auto-synch database
- Authentication with a verified email (with AsyncTask)
- Notification if an item is sold

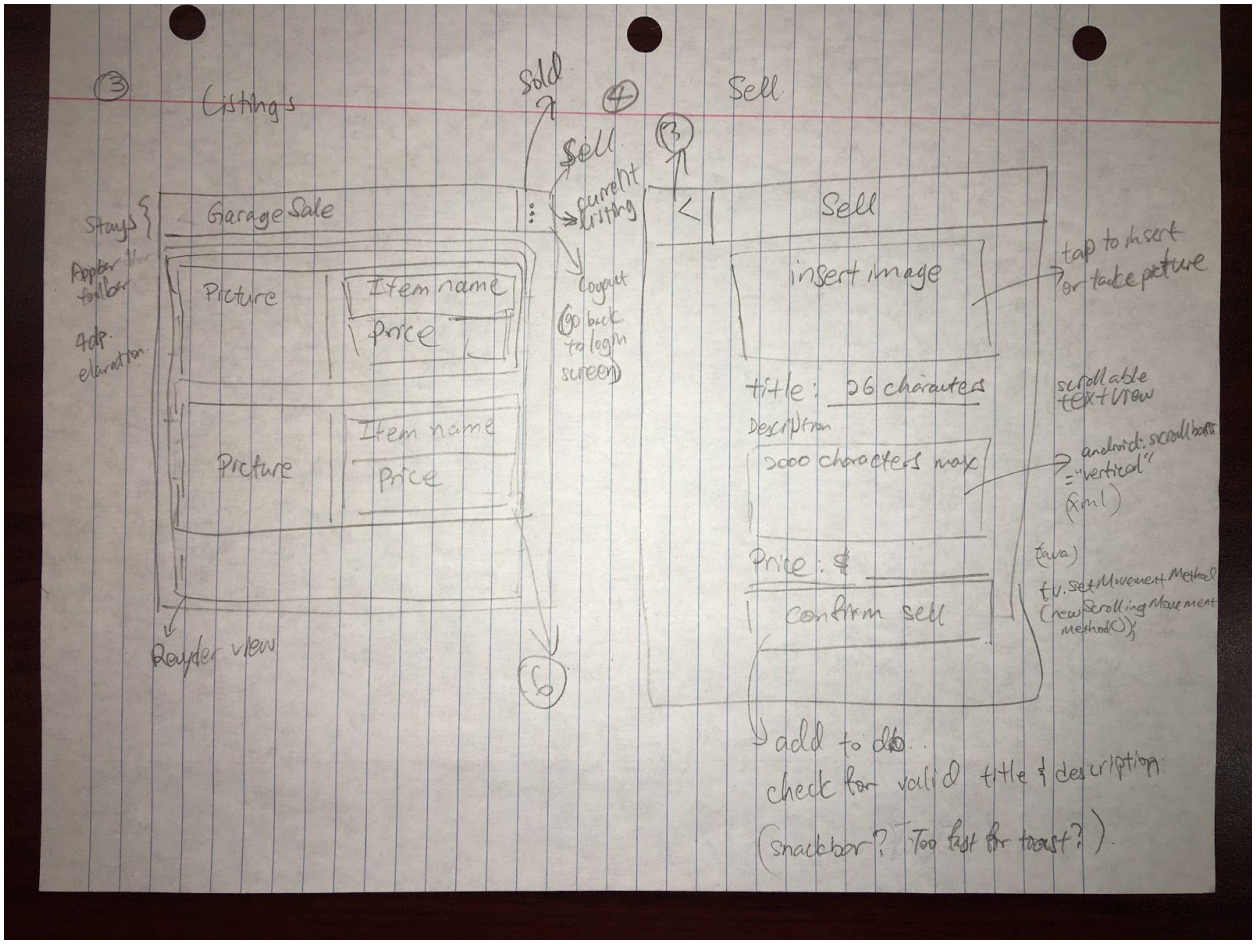
User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Google Drawings, www.ninjamock.com, Paper by 53, Photoshop or Balsamiq.

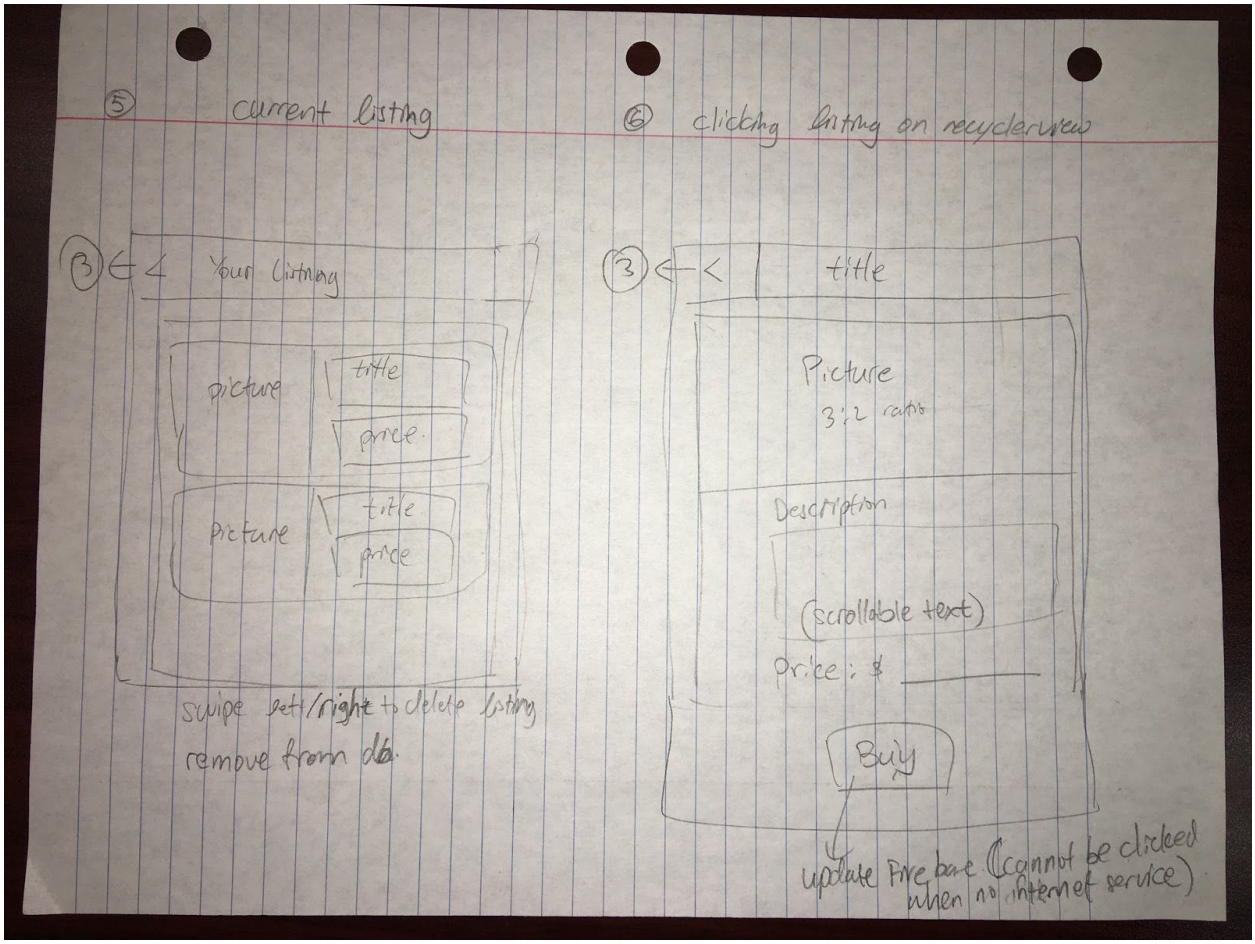
Screen 1



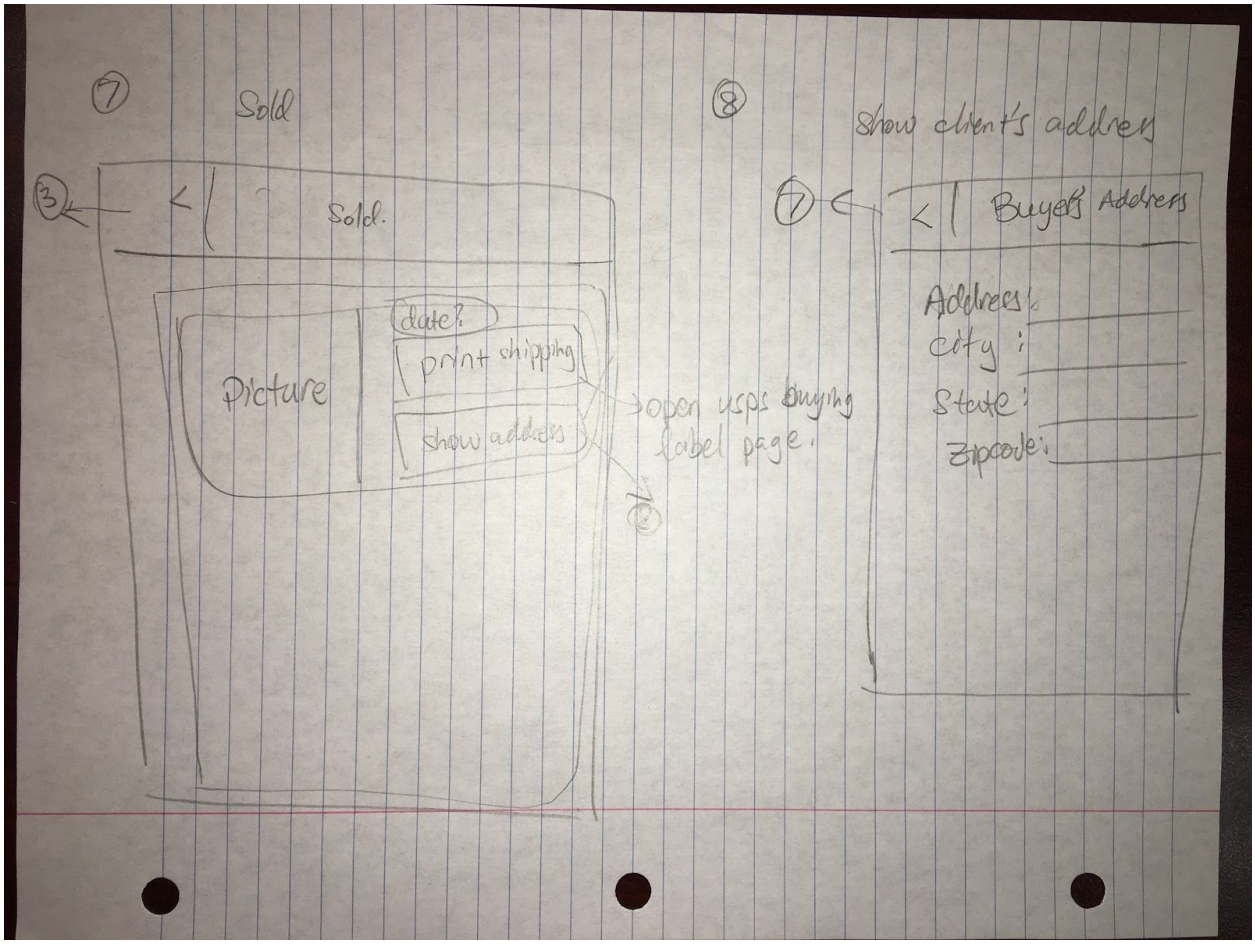
1) Login page, which provides an error message if the password doesn't match with firebase authentication. If the user doesn't exist, prompt a Toast or SnackBar (undecided) for the user to create an account
2) Creating an account by providing a valid email and password. Password has to be retyped and matched. Shipping address needed to know where to ship product from/to. Maybe use for the verified buyer (future implementation). If any of the information is missing, a Toast/SnackBar (undecided) will prompt the first error caught. Reason being to prevent prompting too much error message can overwhelm user experience. Possibly highlight the box that first caught the error.



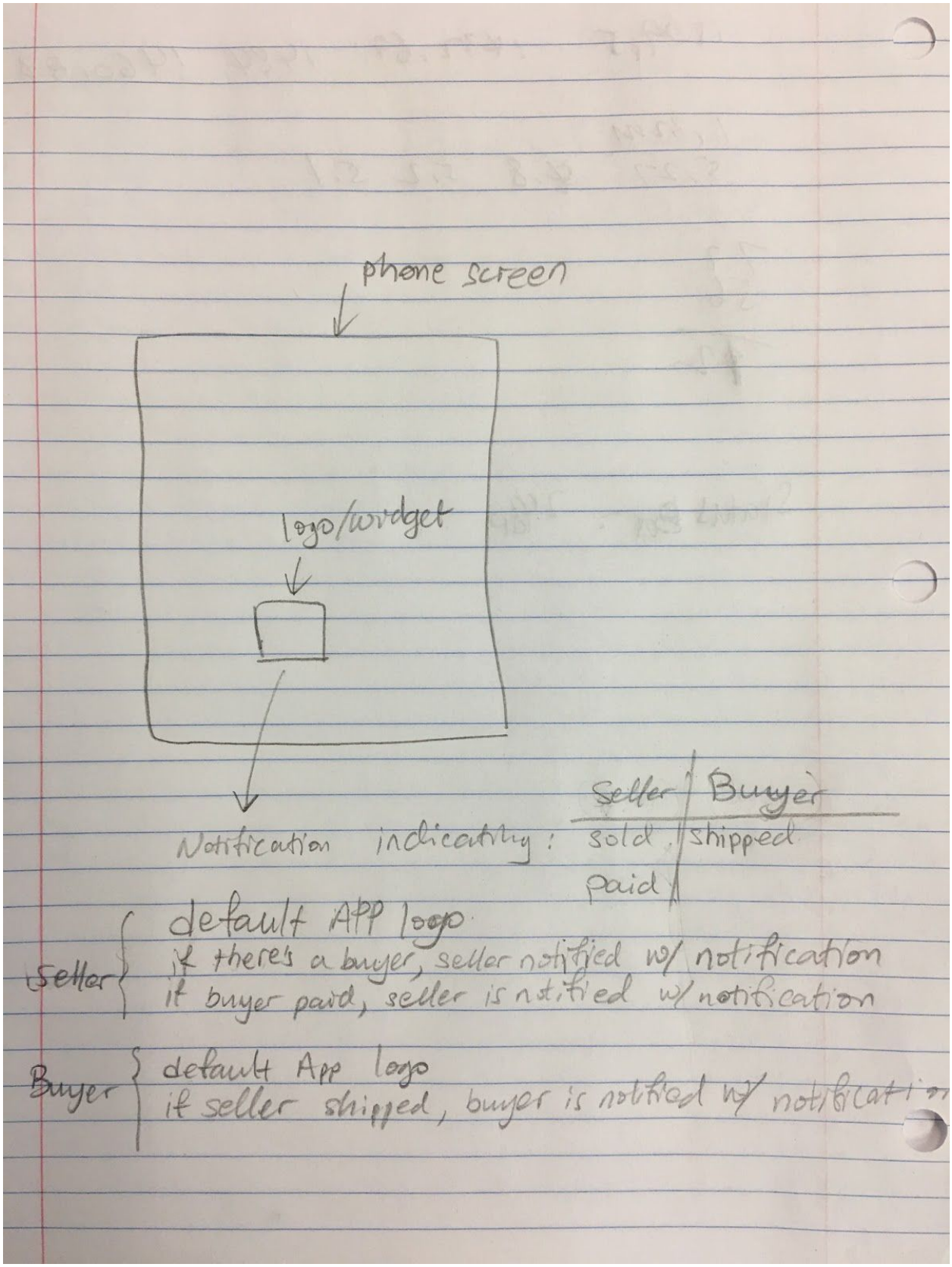
3) This page has the listing that is stored in firebase database. Each item will be displayed on a recyclerview with a picture of the item, the item title, and the price of the item. At the top menu, there will have the option to check the item that the user has sold, if a seller decides to sell, user's current listing, and log out option. Upon clicking either option, the user will be led to corresponding pages.
4) In the sale page, the user can insert an image of the item from photo album or take a picture of the item. The user has to provide item name under "title", item description, and price. Once information has been provided by the user, the user can then click on "confirm sell" and the listing will be shown in (3). Pressing the back button will lead to (3).



5) Current Listing will show the user's current items that are for sale. It will show the pictures, title, and price. The user can edit the listing by selecting the item that the user wants to update. However, if the item is purchased, the user will not be able to edit the listing. The user can swipe right or left to remove the listing from the database. Upon clicking the back button, it will lead to (3).
6) Click listing from (3) will lead to a page with the item's information. The user can see the image of the item, the title, item description, and price before deciding if the user would like to purchase the item. If the user decides to purchase, the user can click on "Buy" and proceed with "Google pay". If there is no internet connection, the buy button will be greyed out or color-coded to indicate that either item is no longer available or no internet connection available. Once user clicks buy, item listing will be removed from listing database, and the seller would be notified.



7) From (3), the user can click on the menu and click on “Sold” to see the items that the user has successfully sold. Upon clicking on “print shipping”, the user will be led to a browser of user’s choice to “USPS” page for the user to buy shipping label. The “Show address” button will show the buyer’s address and will lead to (8).
8) This is the buyer’s shipping address information page. Pressing back will be led to (7)


phone screen
logo/widget
Notification indicating:
Seller
sold
paid
Buyer
shipped
Seller
default APP logo
if there's a buyer, seller notified w/ notification
if buyer paid, seller is notified w/ notification
Buyer
default App logo
if seller shipped, buyer is notified w/ notificati

9) Widget for the seller: default app logo if no one purchases. Widget updates every 30 mins. If there is a buyer and paid before the 30mins update time, “ready to ship” logo will be shown. If the buyer has not paid, then the widget will be shown as “bought” with a notification stating an item was sold but has not been paid.
10) Similar to the seller, the buyer will receive notification if the seller has shipped the product. A notification will be sent to the buyer as well as a change of image.

Key Considerations

How will your app handle data persistence?

Firebase database, firebase authentication

Describe any edge or corner cases in the UX.

Some of the edges are when seller update the item a second before another user click the “Buy” option. I don’t know how fast is the firebase and if it will be able to catch the change. Another is when a seller decides to cancel their listing at the same time as when the buyer deletes the listing. One solution would be giving an hour period for the buyer to decide if they want to cancel, and a refund option for the seller if the item is no longer available.
Also with Google pay, I have not tried using it yet, I do not know how Google pay will show in my app, as this is a mock design. Lastly, when printing the shipping label, the user has to copy the address, which can create typo, while purchasing the shipping label. One solution would be autofill the USPS buying label page, which I am not sure how to do, with the address provided.
Because the Android system enables most of the necessary behaviors by default, the user can navigate using the keyboard. Additionally, “nextFocusFoward” will be implemented for easier accessibility. ContentDescription also will also be implemented. All layouts will be RTL enabled. All strings will be implemented through strings.xml.

Describe any libraries you’ll be using and share your reasoning for including them.

ButterKnife, easy and clean code.

Describe how you will implement Google Play Services or other external services.

Firebase authentication for user signing up and logging in. Unique id created by Firebase can be added to the user’s account, and when a listing is created, the unique id is tied with the listing. When a buyer clicks and purchases an item, the original seller’s unique id can be used to

notify the seller.
Firebase database for listing items and saving user's unique id so a notification can be sent to individual seller when an item is sold.
Google Pay, for accepting and paying the items.

Programming Language.
Java and XML are the main languages for this app.

Stable Libraries, Gradle (androidX), and Android Studio.
Firebase-core:16.0.6
Firebase-auth:16.1.0
(have not decide to use database or firestore)
Firebase-database:16.0.5
Firebase-firestore:17.1.4
Media:1.1.0-alpha01
Legacy-support-v4:1.0.0
Material:1.1.0-alpha02
Appcompat:1.1.0-alpha01
Butterknife:8.8.1 (last known stable version)
Butterknife-compiler:9.0.0-rc1
Junit:4.12
Runner:1.1.1
Espresso-core:3.1.1

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and break them down into tangible technical tasks that you can complete one at a time until you have a finished app.

Task 1: Project Setup

Write out the steps you will take to set up and/or configure this project. See previous implementation guides for an example.

Libraries, Google services, and Firebase:

- Configure libraries (ButterKnife)
- Configure Firebase database
- Configure Firebase authentication
- Configure Google Pay

Task 2: Implement UI for Each Activity and Fragment

UIs:

- Build UI for Sign up
- Build UI for Login
- Make sure firebase authenticate sync with App
- Build UI for AllListing
- Make sure firebase DB sync with App
- Build UI for Sell
- Make sure item is added to the firebase DB
- Build UI for YourListing
- Build UI for ListingItem
- Build UI for Sold
- Build UI for CustomerAddress

Task 3: Check Corner Cases

- Prevent user/seller to change/buy listing while there is no internet
- Remove listing when an item is sold
- Notify specific seller when an item is sold rather than broadcasting to all users

Task 4: Connect with Firebase

Describe the next task. List the subtasks. For example:

- Make sure user login is stored properly
- Make sure listing is stored in the database

Task 5: Listing

Describe the next task. List the subtasks. For example:

- Store user unique id in the listing
- Use the unique listing to notify the seller

Add as many tasks as you need to complete your app.

Submission Instructions

- After you've completed all the sections, download this document as a PDF [File → Download as PDF]
    - Make sure the PDF is named "**Capstone_Stage1.pdf**"
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
- Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"