# College Swap

## Design Document

**Patrick Boatner**

**Charodd Richardson**

**Chris Popovich**

**Justin Kerber**

Change History:

| Date | Summary | Author(s) |
|---|---|---|
| 1/22/2015 | Start of doc. Outline of requirements and use cases. | Team |
| 1/29/2015 | Draft of introduction, description, and requirements. | Team |
| 1/29/2015 | Draft of class diagram. | Patrick Boatner |
| 2/03/2015 | Draft of activity and use-case diagrams. Refined class diagram. | Team |
| 2/05/2015 | Finished activity diagrams. | Chris, Charodd |
| 2/05/2015 | UML class diagram | Patrick Boatner |
| 2/05/2015 | Use case diagram and descriptions | Justin Kerber |
| 2/17/2015 | Class diagram description | Patrick Boatner |
| 3/3/2015 | Updated use case diagram | Justin Kerber |
| 3/3/2015 | Added Class diagram relationships | Patrick Boatner |
| 3/6/2015 | Created Login, ViewProfile sequence diagrams | Charodd Richardson |
| 3/7/2015 | Created ViewListing, corrected sequence diagrams | Chris Popovich |
| 3/8/2015 | Expanded class diagram | Chris, Patrick |
| 3/8/2015 | Sequence diagram descriptions | Chris, Patrick |
| 3/8/2015 | Update design document and presentation | Chris, Patrick |
| 3/8/2015 | Created create new and update listings sequence diagrams | Justin Kerber |
| 3/10/2015 | Update sequence diagrams, use case, and class diagram | Team |
| 3/12/2015 | Create “Design” section with changes, clean up document, and finalize presentation | Patrick Boatner |

## Table of Contents

# 1. Introduction

## 1.1 Motivation/Purpose

College Swap is an Android application that will serve as a platform, much like Craigslist, where college students can exchange goods and services with other students. The motivation behind this project is that, as college students, we often find ourselves in need of an easy and secure way to sell and buy things with fellow students. The main item we find ourselves in need of exchanging is textbooks for school which is the main motivation behind this project.

### 1.2 Scope

Initially this app will be made to serve the University of Alabama but could later be expanded to work with any other colleges. The users who would be able to access the application would be limited to those who have a 'crimson.ua.edu' email address in this version although if expanded to other colleges we would add support to allow those users as well. The app will not have support for purchasing through the app but will instead encourage the buyer/seller to meet at a safe location on campus. The app will be bounded to cell phones and will not be available for use on a tablet. However, this could in the future be an added feature.

### 1.3 Goals

The goals of College Swap are to provide a user friendly and secure platform for college students to trade textbooks, football tickets, and subleasing apartments with their peers.

### 1.4 Key Definitions

Metadata: Data about data.
Listing: A collection of information describing a good or service that a student wants to sell or rent. It includes data like date, title, description, and various fields related to the type of listing.
Transaction: A successful sale or rent from one student to another, which was facilitated by a listing.
Poster: The student who creates or 'posts' a listing so other students can view it.

# 2. Project Description

### 2.1 Selling

The app will provide a reasonable interface for entering information on items users want to sell. The users will be able to enter information such as the item description, category, and price. The app will also allow users to select their preferred contact method for the item. Once the information has been posted, users will be able to view the current offers or remove the item from listings. Once the item has been sold, the seller will be able to rate and post a review on the buyer.

### 2.2 Buying

The app will allow buying users to view, filter, and sort items by information such as category, price, and seller history. During searches, the items will be displayed in a list with their basic information. Selecting an item displays the detailed description of the item, as well as seller information and ratings. Buyers will be able to rate and review the seller of an item they buy.

### 2.3 Users

The app will require users to log in with a crimson.ua.edu email, which will be verified during acount creation. Users will have preferred settings for contact information and alerts. The server will also collect and make available buyer/seller ratings and reviews.

# 3. Requirements

## 3.1 Functional

The app should provide an account for each user. The app must use a “crimson.ua.edu” email address to verify the student is from UA. The app must also allow each user to configure settings for their account, which at least includes contact settings. The user can choose whether others can contact them regarding their listings via phone, push notification, or email.

The app should allow users to rate transactions with other users. It should also allow users to view these transaction and rating histories for another given user.

The app should allow users to post listings to sell their textbooks, football tickets, and subleases. It should allow users to view these listings from other users. It should allow the user to view a summary page of multiple listings and a detailed page of a single listing. It should allow users to sort the summary page by date and popularity.

The app should allow users to contact the poster in order to purchase the listing.

The app should provide different categories of listings. The app should include fields specific to each category and make use of these fields whenever relevant, including posting a listing and searching for listings.

The app should allow users to search within a category for listings whose fields match certain criteria.

The app should provide push notification support to users. Users should be able to receive push notifications when others want to buy or make an offer for their listing. Users should be able to contact other users using push notifications. Users should be able to register for push notifications whenever listings matching certain criteria become available.

## 3.2 Non-Functional

The app should provide security and confidentiality for contact information.

The app should provide a reasonably responsive user interface.

The app should be portable across a large variety of android phones.

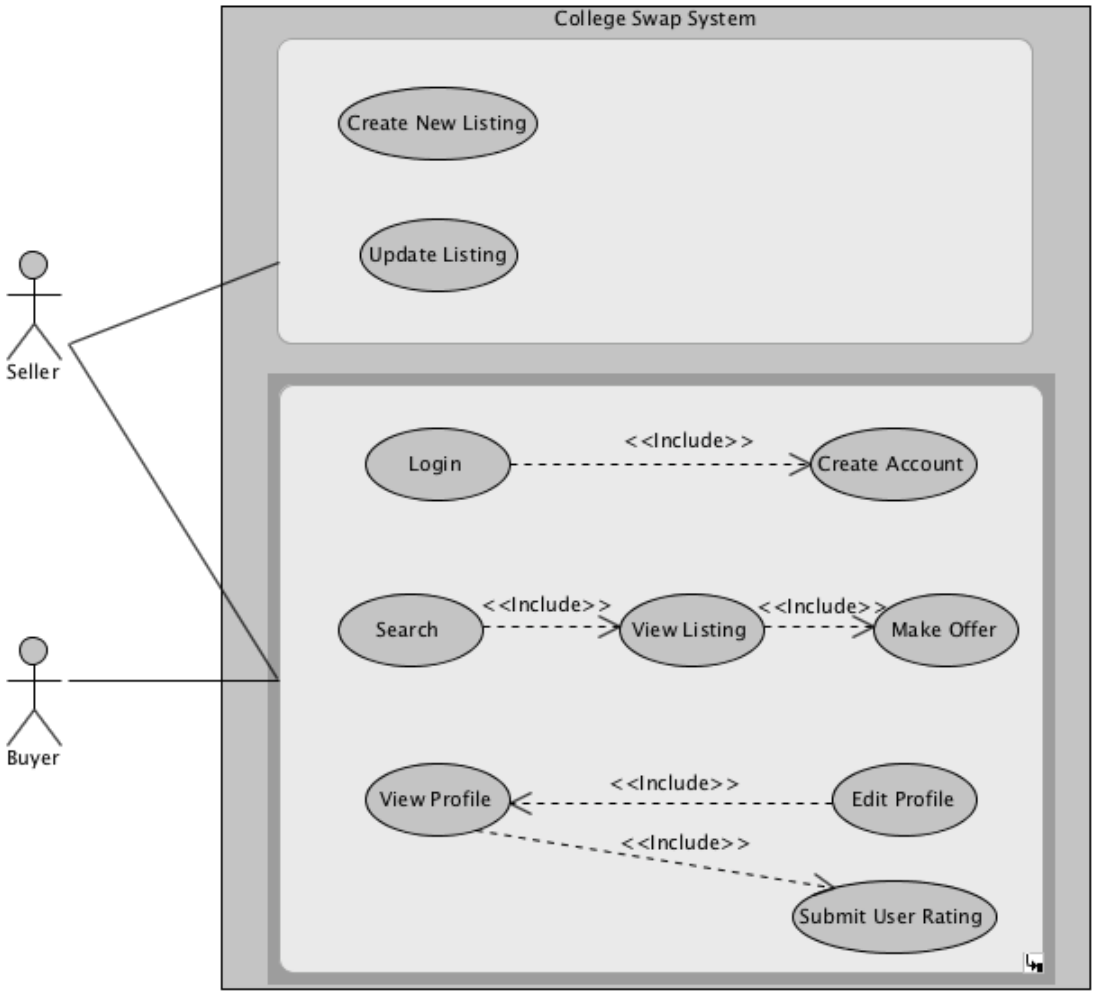
# 4. Requirement Diagrams

### 4.1 Use Cases

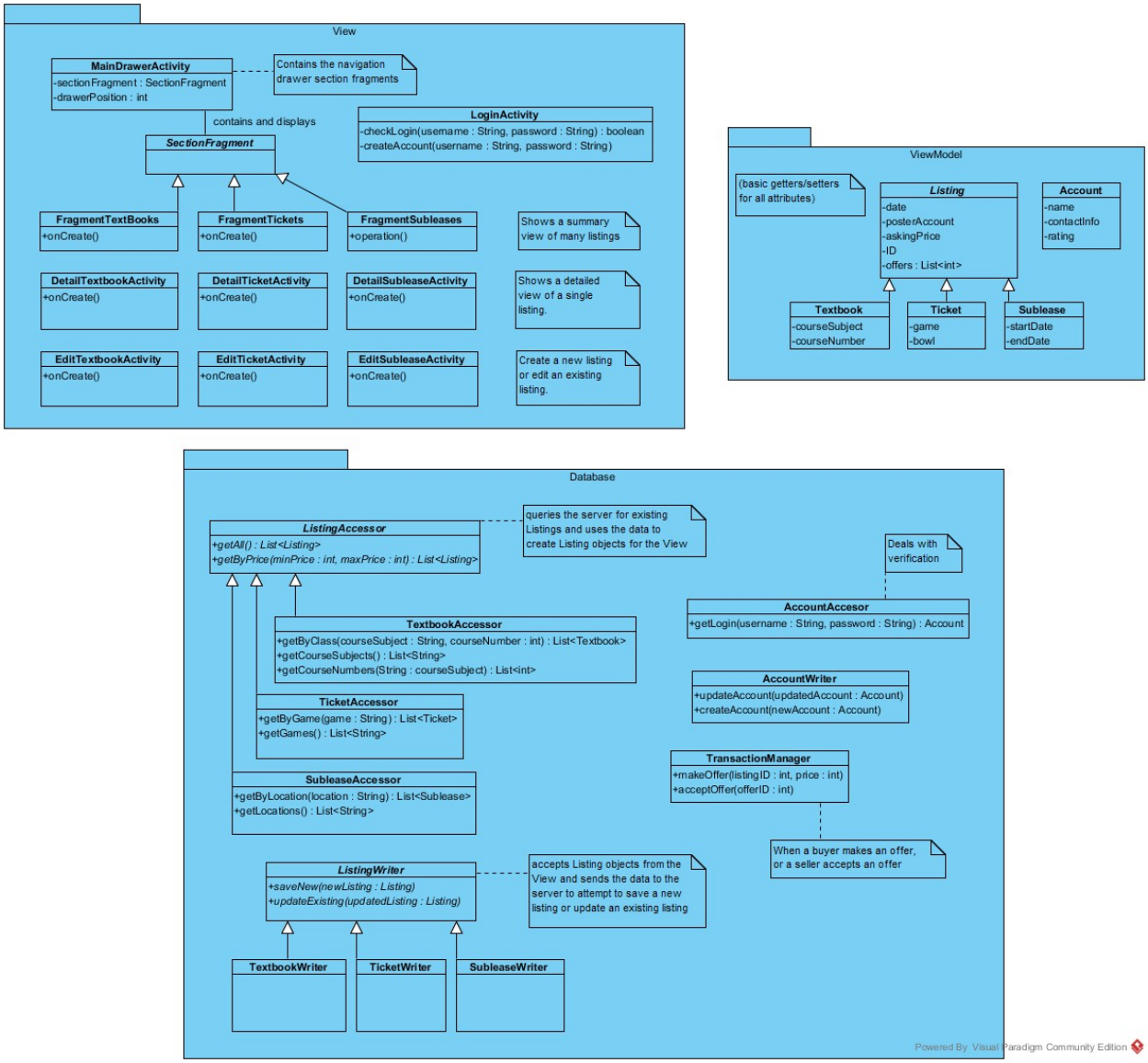
Users can log in to the College Swap app by entering their email and password. If the user does not have an account, they can create one by entering their email and password.

Once logged in, users can search the listings. They first select which category (Textbook, Ticket, Sublease) they wish to search, then add search criteria (price, textbook subject). The app provides the user with a list of listings matching the criteria. The user can then click on a listing to open a detailed description of the listing. While the user is viewing a listing, they can enter a price and make an offer on it.

Sellers can create new listings and update previously created listings. The app shows text fields of the listing attributes and allows the seller to modify and save changes.



## 4.2 High-Level Class Diagram




Our class diagram consists of three packages- the View, ViewModel, and Database package. The View package consists of the classes Android needs to display activities. The ViewModel package consists of classes which provide the information needed by the View package. The Database package is responsible for generating and updating ViewModel objects for the View. We are trying to follow the Model-View-ViewModel architecture pattern, using Database as the Model.

The View package has largely been determined by the Android APIs. We will use a navigation drawer layout for switching between different types of listings. The navigation drawer layout needs a main activity to host the various sections (or pages), and MainDrawerActivity fulfills this purpose. The sections hosted by MainDrawerActivity are not Activities, but Fragments. We will call them SectionFragments. This allows Android to switch

between sections without reloading the whole Activity. There are three subclasses of SectionFragments- FragmentTextbooks, FragmentTickets, and FragmentSubleases. These Fragments show a summary list view of many listings. They each have an onCreate() method like an Activity, allowing them to perform initial setup when loaded. For each type of listing, there is also a corresponding Activity to view the details of a single listing and to edit a single listing. The edit Activities will also allow us to create a new listing.

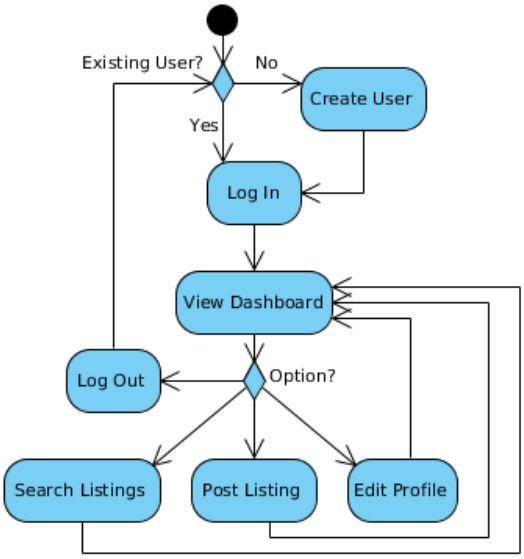
The ViewModel package consists of Plain Old Java Objects (POJOs). They don't extend any class, and they only provide getters and setters. These classes model various types of data, such as Listings and Accounts. They have fields to store any kind of data needed to display and store the listing. Listing subclasses have fields relevant to the specific type of Listing, such as course number or football game. Once the View has a reference to a ViewModel class, the View can simply call the ViewModel getters and do whatever Android requires to display the data.

The Database package is responsible for generating ViewModel objects containing previously stored data. Accessor classes allow accessing the data by generating ViewModel objects. Writer classes allow writing or storing the data by accepting ViewModel objects and saving their data to the database. There are subclasses of ListingAccessor and ListingWriter for the three types of Listings. The ListingAccessor subclasses provide methods to get Listings filtered by fields specific to that type of Listing. For example, the TicketAccessor provides a method to get Ticket listings for a certain game. The AccountAccessor class handles verification, and the AccountWriter class allows updating user preferences and creating new accounts. The TransactionManager handles making and accepting offers. By defining these database classes early in the development process, we can quickly define these methods to return ViewModel objects with mock data. This will allow us to simultaneously begin working on the Android View and the server side component. Once we are ready to integrate the app with the server, we can change the content of these methods without affecting the rest of the app.

## 4.3 Activity Diagrams

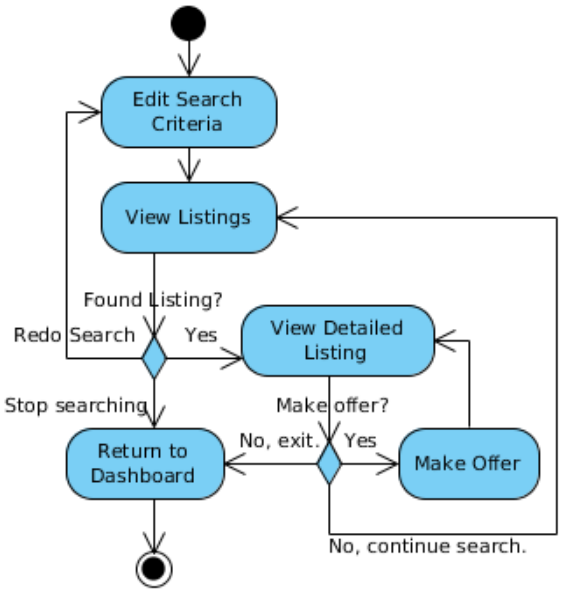
### 4.3.1 Log In

When the user starts the app, they have the option to either log in as an existing user or create a new user. After logging in, the user is directed to their dashboard. At this point, the user can select from several options: Search Listings, Post Listing, Edit Profile, or Log Out.



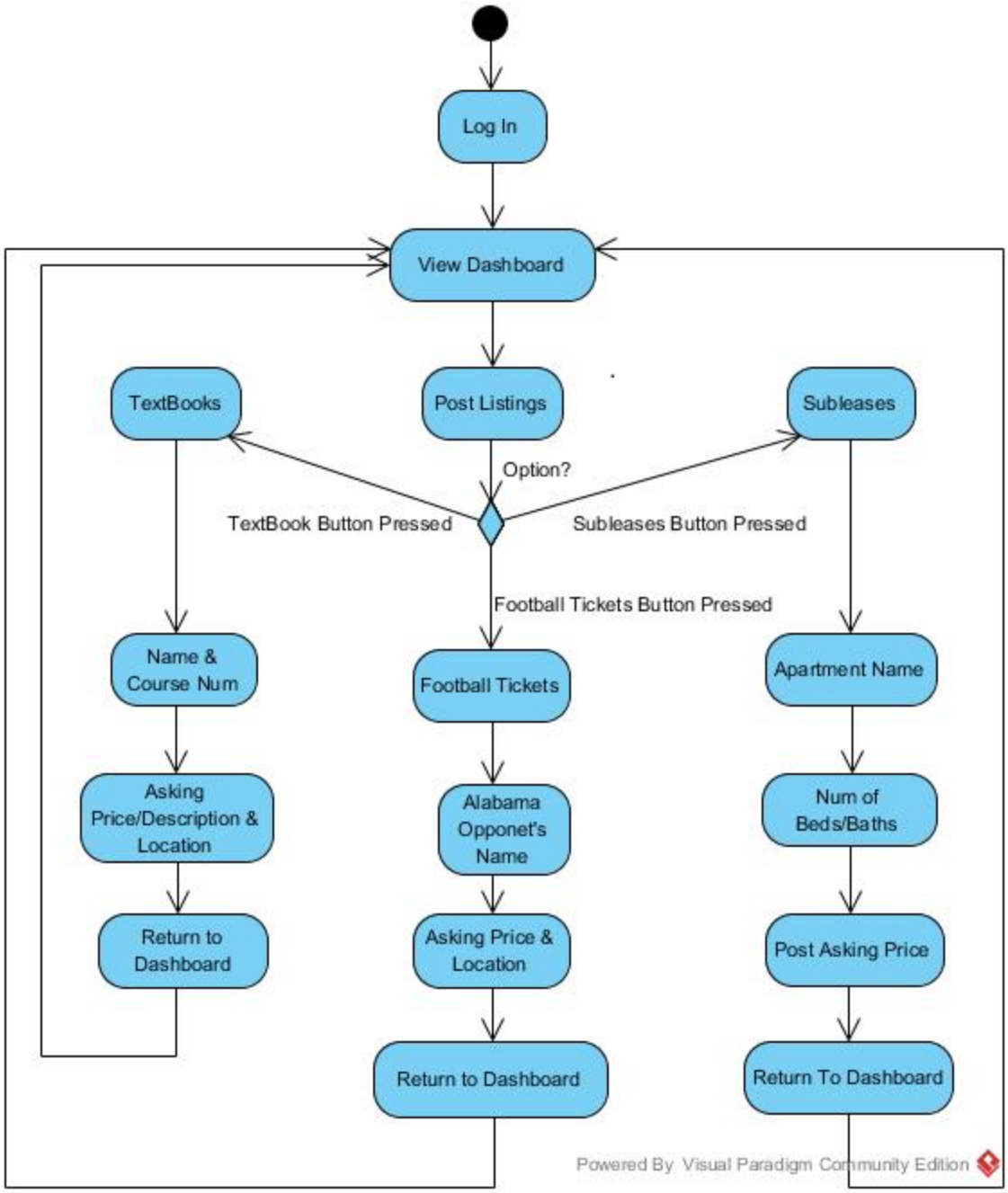

### 4.3.2 Search Listings

After choosing to search listings from the dashboard, the user enters criteria (price, textbook subject, etc.) to search for. The app displays the listings that fit the criteria. The user can select a listing, redo the criteria, or go back to the dashboard. If the user selects a listing, the app displays a detailed view of that listing. If the user wants the item, they can make an offer at this point. Otherwise, they can either go back to the search results or quit searching.



### 4.3.3 Post Listing

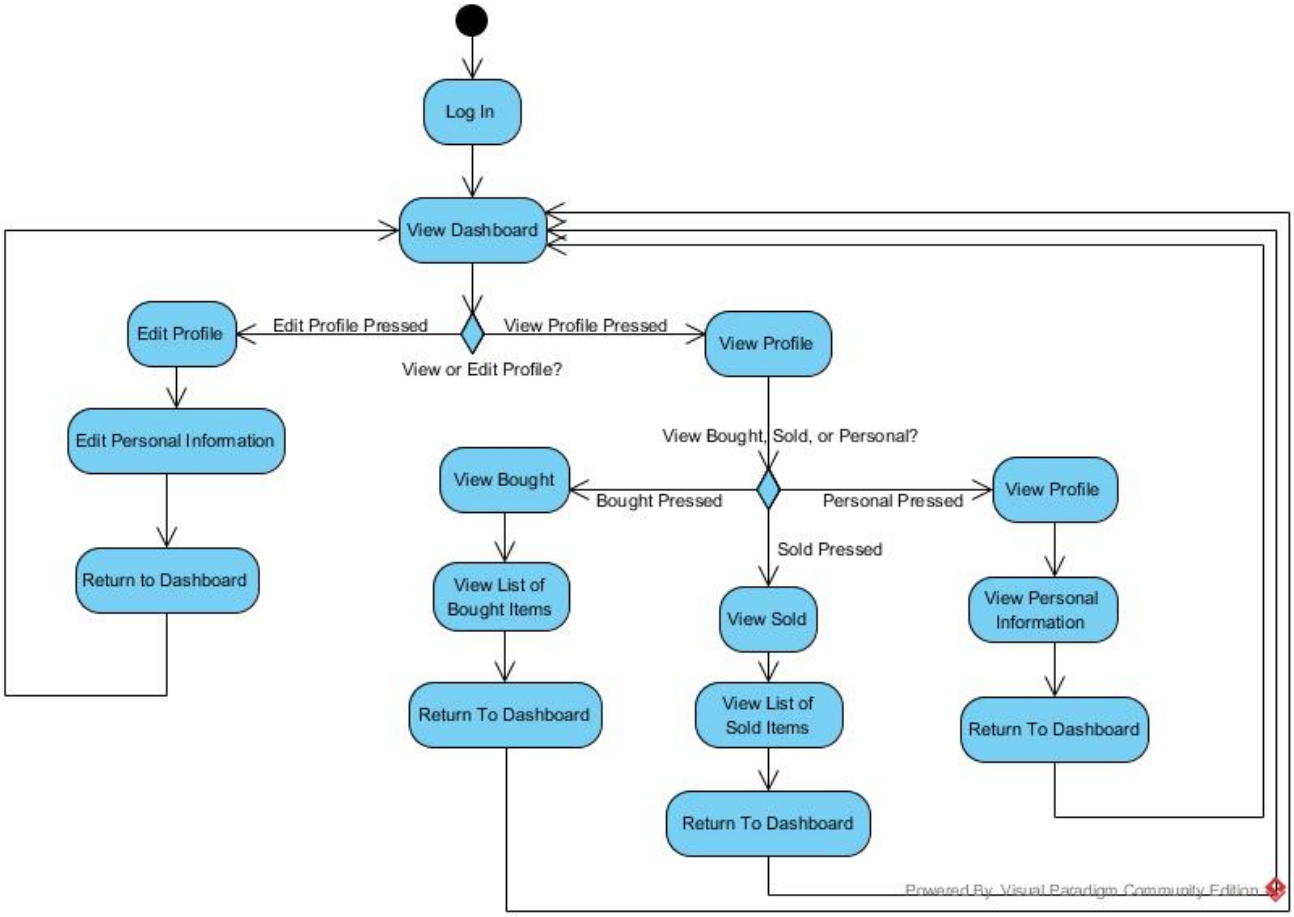
After the user logs in, they can choose to post a listing. The user is asked to indicate what type of listing they wish to post (Textbook, Ticket, Sublease). The app requests that the user input the relevant information for the listing. The user enters the information and submits the post. The app then returns to the dashboard.



### 4.3.4 View/Edit Profile

After the user has logged in, they can choose to view or edit their profile. If the user chooses to edit their profile, the app displays their personal information in editable text fields. The user edits the information and then returns to the dashboard.

If the user chooses to view their profile, they can choose to view the listings they bought or sold, or view their personal information. The app displays the requested information. When the user is done, they return to the dashboard.



# 5. Design

### 5.1 Scope Changes

The scope of College Swap has not changed a great deal. It will no longer allow viewing the full profile of other users. After further consideration and planning the interface, we believe relevant information will be shown in other places (such as purchase history), so this will no longer be necessary.

Text search will also be outside the scope of this system. This decision was driven by a need to trim back the size of the system. Users will still be able to choose the category of listings, then filter by fields relevant to the category. We believe that these filtering options will be more than sufficient in lieu of a search feature.

### 5.2 Feature Changes

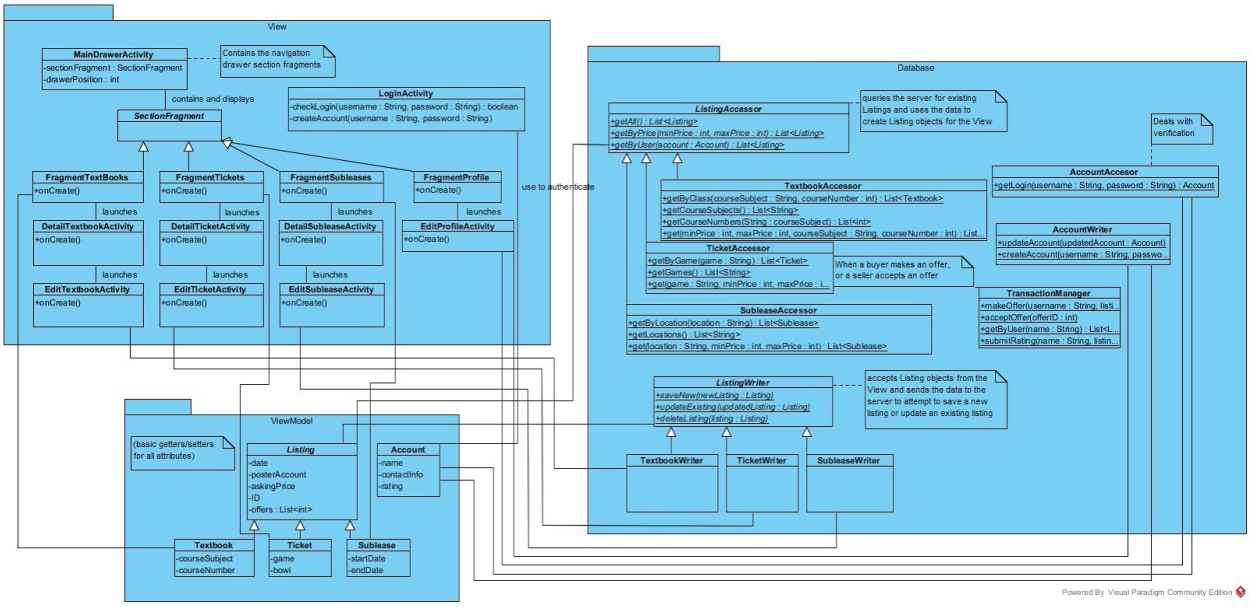
We initially planned to offer users multiple notification options (such as when another user makes an offer on their listing). Instead, we will only support email notifications. This will eliminate the need to interface with Google's Cloud Messaging service. This will also eliminate the need to interface with a text messaging service. Although these would be nice features to provide in the future, there is likely not enough development time.

# 6. Design Diagrams

## 6.1 Detailed Class Diagram

Our high level class diagram already had a lot of detail, so there haven't been many changes. In the View package, we have added a Section Fragment to display the user's profile, plus an Activity to edit the profile. This Section Fragment will be responsible for showing the user their transaction history.

The Database package contains the majority of the changes. The ListingAccessor now provides a method getByUser() to retrieve all the listings for a given user. These will be displayed on the user's profile. There are also methods for each subclass of ListingAccessor to retrieve Listings which are both within a price range and meet other criteria. The TransactionManager also provides methods to get transactions by user, and to submit a rating for a transaction.



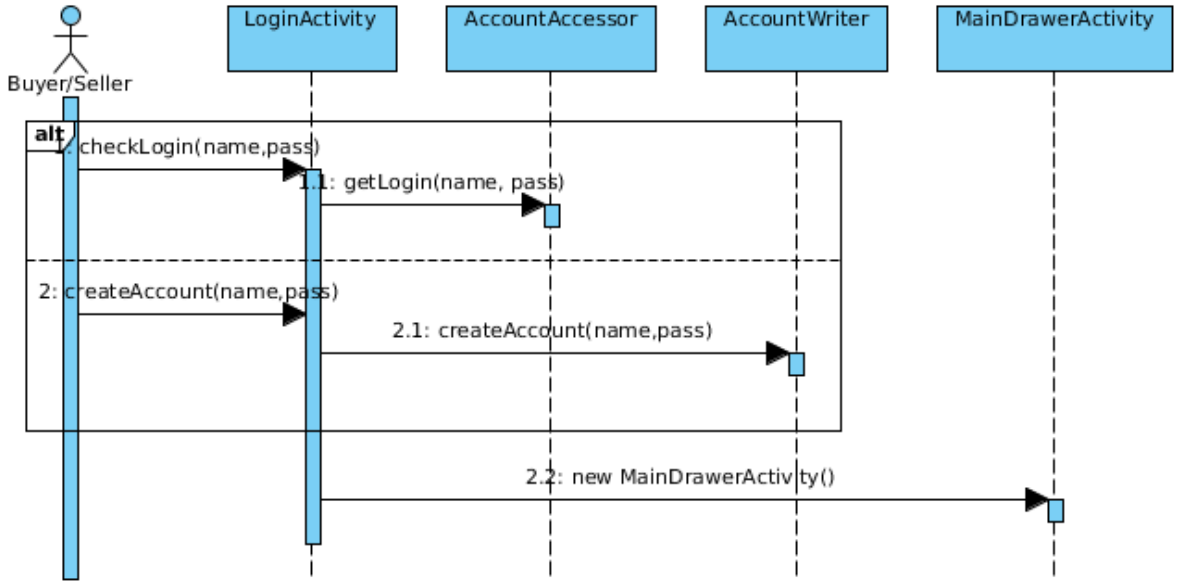
## 6.2 Sequence Diagrams

### 6.2.1 Log in / Create user

In this sequence diagram, the user logs in. If the user has an account, they submit their name and password through LoginActivity.checkLogin(name,pass). LoginActivity sends the name and password to AccountAccessor via getLogin(name,pass), which verifies and returns the associated account information.

If the user does not have an account, they also enter their name and password. They use LoginActivity.createAccount(name,pass), and LoginActivity informs AccountWriter to create an account with createAccount(name,password). LoginActivity then proceeds with a blank account.

Once LoginActivity has an account, it creates a MainDrawerActivity. The user has successfully logged in.



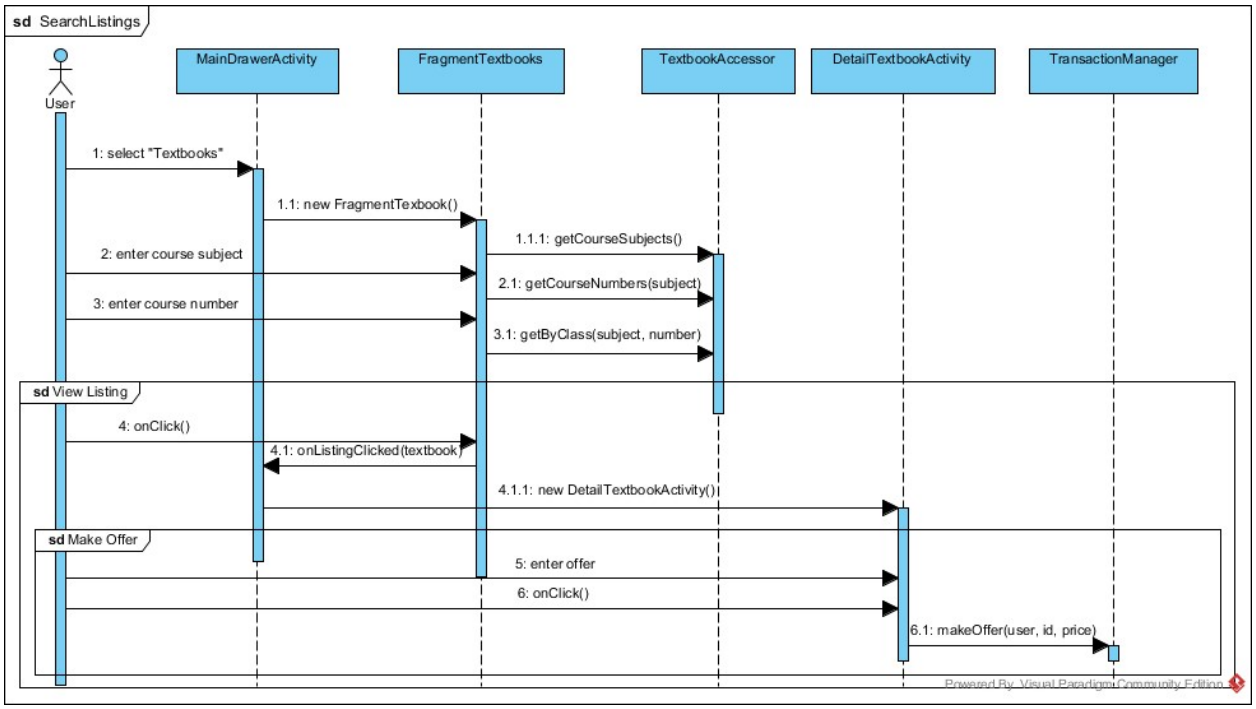
### 6.2.2 Search listings / View listing / Make offer

In this sequence diagram, the user searches for Listings. To begin, they choose a category in the navigation drawer. For example, suppose they choose “Textbooks”. Other Listing categories have the same method calls. The MainDrawerActivity creates a new FragmentTextbooks. The user enters criteria for the Listings they want to see. The FragmentTextbooks then calls the appropriate method on TextbookAccessor to retrieve the Textbooks to show. For example, if the user is just filtering by class, then FragmentTextbooks will call TextbookAccessor.getByClass() to retrieve the list of Textbooks to display.

The call to TextbookAccessor.getCourseSubjects() gives the UI a list of subjects needed to populate a menu. Once the user chooses a course subject, the call the TextbookAccessor.getCourseNumbers(subject) gives the UI a list of course numbers for the chosen subject. These method calls enable a core feature of College Swap. By allowing students to choose from a list of values for various fields, we ensure that searching for a specific listing is easy and accurate.

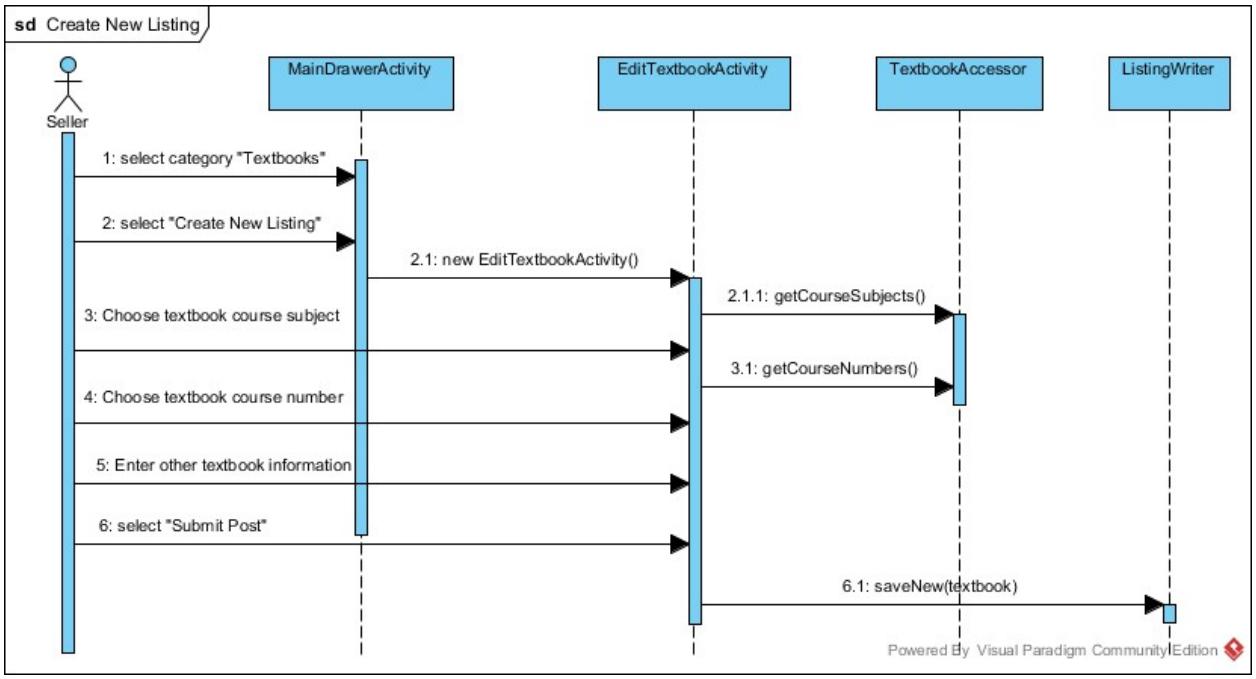
The user may then view an individual listing by clicking on it. When they click on a listing, the FragmentTextbooks calls onListingClicked() on MainDrawerActivity so it can launch a new Activity. The MainDrawerActivity will launch a new DetailTextbookActivity with the Textbook object that was clicked. This Activity displays information about the individual Textbook listing, including the current offer price.

The user may then make a new offer on the Textbook listing. They enter their offer into an input box, then click “Make Offer”. DetailTextbookActivity then calls TransactionManager.makeOffer() with the details of the offer they just made. This sends the offer to the server. When this completes, the DetailTextbookActivity updates itself to reflect the new offer.



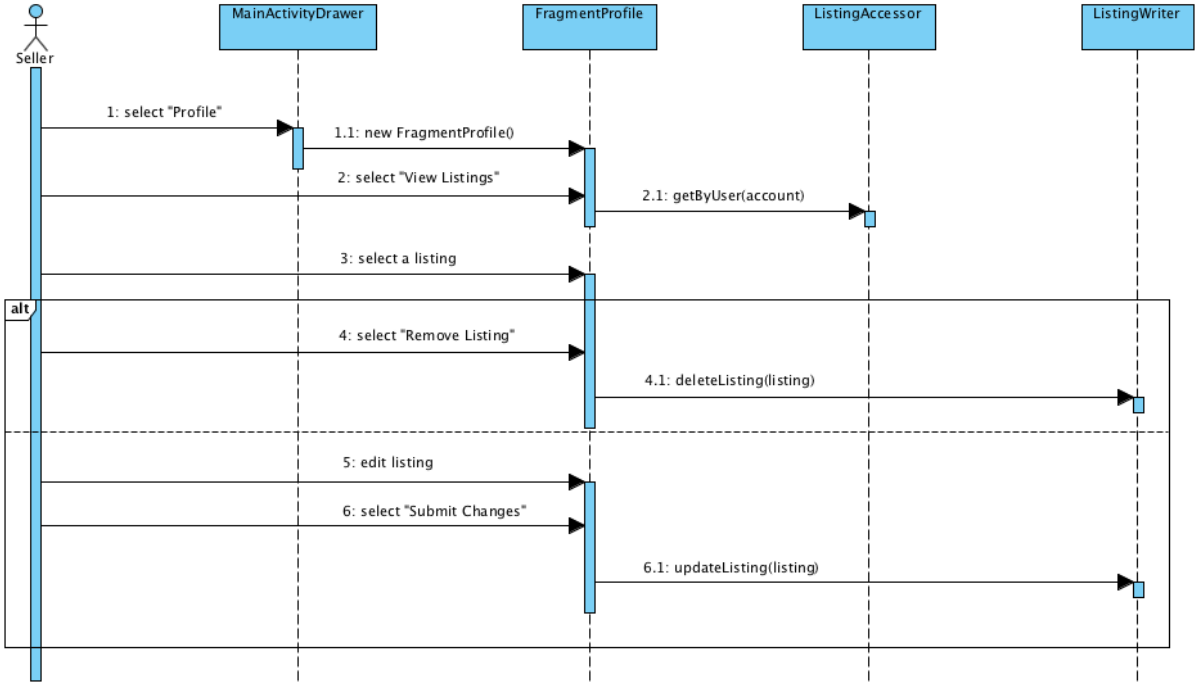
### 6.2.3 Post New listing

In this sequence diagram, the user creates a new listing. They have already chosen a category of Listings in the navigation drawer. This diagram assumes a Textbook Listing is being created, but the diagrams will be the same for other subclasses of listings. When they click the “Create New Listing Button”, a new EditTextbookActivity is launched. This Activity has input fields for the specific type of Listing (in this case, a Textbook). The users enters the information for the Listing they want to post, then clicks “Submit Post”. This calls ListingWriter.saveNew() with the new Textbook (or other Listing) object. The ListingWriter then submits the fields of this object to the server. When this completes, the Activity closes.



### 6.2.4 Update Existing Listing

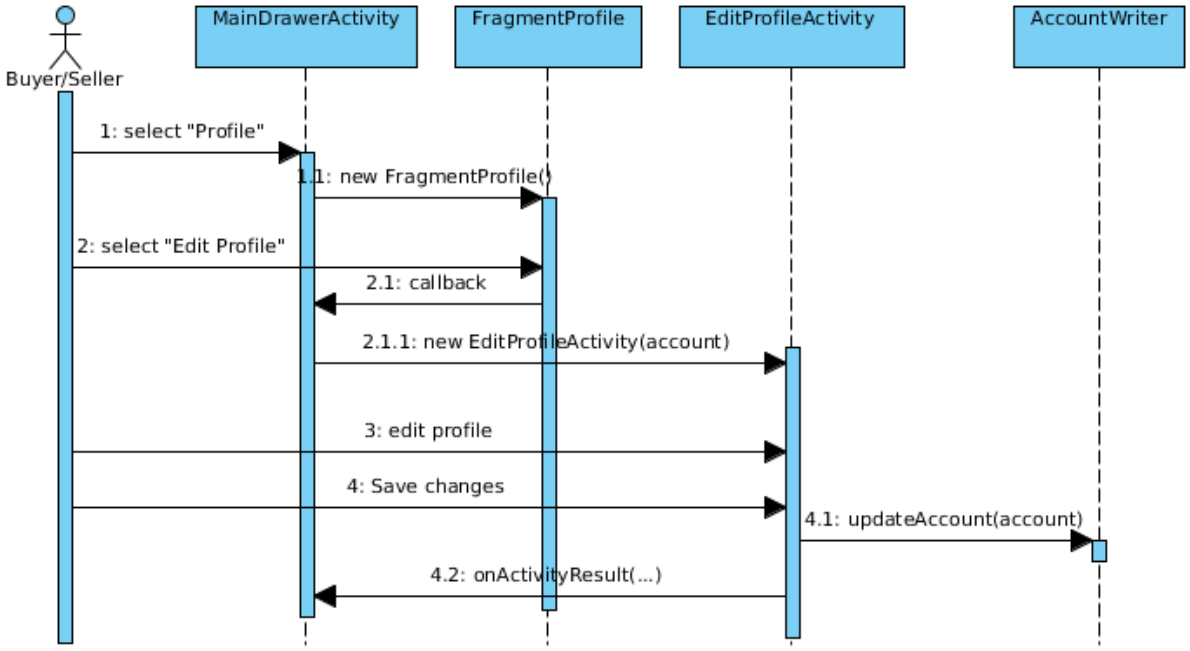
In this sequence diagram, the user is updating an existing listing that he/she has posted. Assuming the user has already logged in they will select to view their profile which will launch a new FragmentProfile. The user will select to view their listings and a list of all the users posts will be queried from the database and displayed on the screen. The user will then select a specific listing which will open the detailed view of the listing. From this screen the user can either delete the listing which will call deleteListing() from the ListingWriter object. Otherwise the user will edit the listing and submit the changes which will call updateListing() from the ListingWriter object.



### 6.2.5 View profile / Edit profile

In this sequence diagram, the user views their profile. They begin by selecting “Profile” in the navigation drawer. The MainActivityDrawer creates a new child FragmentProfile to display their profile. This allows the user to see details about their own profile, such as Name and contact information.

If the user clicks “Edit Profile”, the FragmentProfile makes a callback to the parent Activity in order to launch an EditProfileActivity. The parent Activity launches a new EditProfileActivity with the user’s account. The EditProfileActivity is similar to the FragmentProfile to view the profile, but most of the fields are editable. After the user has edited their profile, they click “Save Changes”. This calls AccountWriter.updateAccount() with the updated account object. The AccountWriter will submit this change to the server. When this completes, the Activity ends and Android calls MainDrawerActivity.onActivityResult(). This lets the MainDrawerActivity know that the account has been updated and it needs to reload the information.



### 6.2.6 Submit user rating

In this sequence diagram, the user adds a rating to another user's account. First, the user opens their profile page (the FragmentProfile class) from the MainDrawerActivity and clicks on the "Transactions" button. FragmentProfile uses TransactionManager's getByUser(name) method to collect and display the transactions the current user has made. The user selects a transaction and enters a numerical rating. FragmentProfile then calls TransactionManager.submitRating(name, listing, rating) to submit the rating.