Mooch

Project Plan
Fall 2016

Whirlpool Contacts:

Jeff Stoller

Mega Agrawal

Vikram Bharadwaj

Michigan State University Capstone Members:

Noah Hines

Daniel Jiang

Laura Robb

Adam Schoonmaker

Caleb Swanson

Table of Contents

Executive Summary

As the top appliance manufacture, Whirlpool Corporation has a vested interested in simplifying consumers' lives. One of the areas in the home that invokes excitement and passion is the kitchen. The US food economy, which includes everything from grocery stores to fast food, is a 1 trillion-dollar market - half of which is attributed to eating at home. With the brand portfolio of Whirlpool, KitchenAid, Maytag, Amana and more, Whirlpool Corporation has the ability to enhance the in-home food journey.

An idea was recently proposed to focus on an opportunity to utilize the "sharing economy." Imagine you live in a large apartment complex in Chicago. Food is all around you, but access to a grocery store for a quick purchase can be a challenge with the high traffic volumes. What if you were baking and didn't realize you were out of vanilla? What if you got home from work and you were craving a snack you didn't have in your apartment, like soda or candy? These are the problems that Mooch will help solve.

The purpose of the capstone project is to build a Proof of Concept mobile application that will enable people to find ingredients and food items from others within their apartment or condominium community. There are people that are willing to trade, sell or giveaway their food, and there are people who would like to purchase or take food from other users. The application enables members of a community to avoid lengthy trips to the store for a small purchase, cut down on food waste, and facilitate social interaction between neighbors.

Functional Specifications

The primary goal of this project is to create a Proof of Concept of the Mooch application that will allow users to search for available food within their apartment or condominium community. The application will let users post listings of food they have that they would like to sell or share. This will encourage social interaction between members of a housing community as well as help reduce food waste.

The secondary goal of the project is to develop a web application that will be used to track various activities of the mobile applications. Members of the Mooch team can use this application to manage communities and view activity of the mobile application.

Mobile Application

Members within apartment and condominium communities will use the mobile application to search for available food items and list their own items to sell, trade, or give away. It will be similar in idea to applications used to buy and sell goods, such as *letgo*.

Users can browse listings with or without an account. When launching the application for the first time, users will have to select the community to which they belong. After that, they will be able to see listings other users have posted in that community, and a logged in user will be able to add new listings. A user's community can be changed at any time.

Listings will contain information about the food that is being sold, including price, description, picture, and category. The item categories are similar to isles in a grocery store, such as Produce, Deli, or Frozen foods.

After a listing is created, buyers can put in a request to purchase the item. After the seller accepts one of the requests, the buyer will have access to the seller’s contact information and can initiate a discussion on completing the transaction in person. Each user will determine the contact information he or she wishes to share with buyers, such as email, phone, or apartment number. Only one form of contact information will be necessary.

Web Application

The web application will be an administrative tool used by the Mooch team. This application will allow those with administrator access to add or remove listings, users, and communities from the database. They will also be able to modify the community records. The administrators can also view statistical data about how many users and transactions there are, which can be filtered by daily or total values. This will allow the Mooch team to see how much activity has occurred each day in comparison to the entire history of the mobile application.

Design Specifications

As a consumer-facing mobile application, a simple, refined, and attractive user experience is key to the success of Mooch. Consumers have high expectations of their apps and if Mooch does not meet them it is unlikely they would trust using it to exchange food with other people. This section contains screen mockups and example use cases.

User Interface

Mobile Application

Initial Loading

The initial loading screen is the screen a user sees when the app launches. The first time the app opens after being downloaded, the app prompts the user whether they would like to subscribe to push notifications. There is a spinner that spins to indicate loading. While the loading screen is displayed, the app checks if there is any saved user data. If there is saved user data then the app attempts to automatically log the user in. The app also downloads the available communities and food categories from the API, and updates the user's push notification device identifier if they are subscribed. When these steps are completed the Initial Loading screen fades away. If a user was successfully logged in they are taken to the Listings screen. If a user was not logged in, they are taken to the Community Picker.

***Figure 1**. Initial Loading*

Community Picker

The Community Picker shows a user the communities that they can join. Communities are displayed as a collection of photos with the community name as a caption. They can be sorted alphabetically, or by the distance from the user's current location. When the user selects a community the screen is dismissed and the user is returned to the screen they came from. However, if the user came from the Initial Loading screen, then they are taken to the Listings screen.

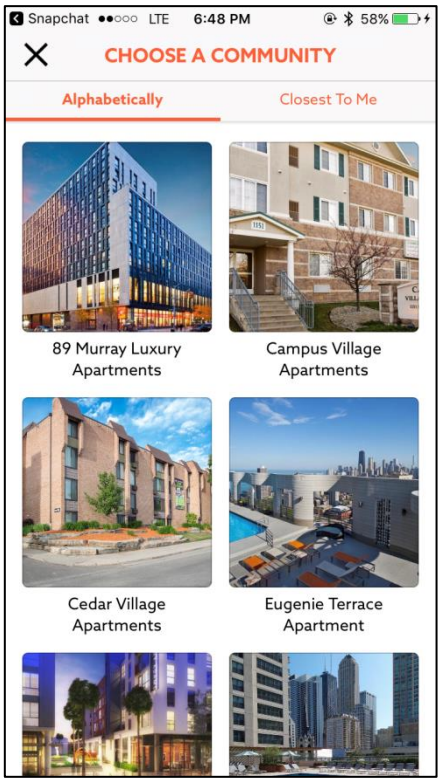


***Figure 2a**. Community Picker, alphabetically sorted*

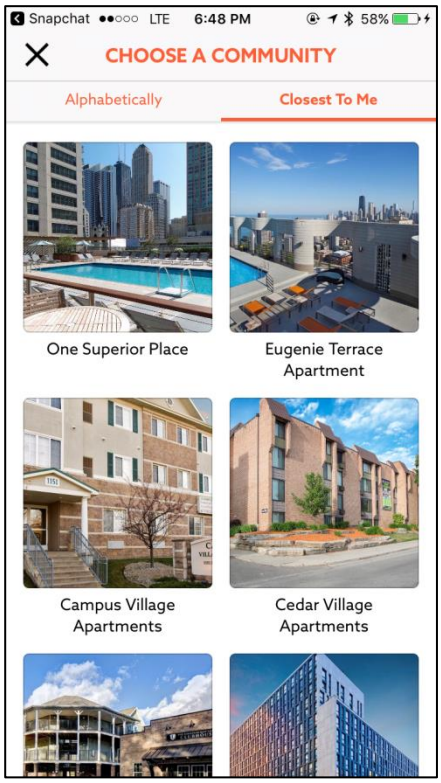


***Figure 2b**. Community Picker, sorted by nearest location*

Main Navigation

Once the initial community has been selected, the main interface of the app becomes visible. The navigation of the app is based on a pattern common to iOS applications, the tab bar. Each icon indicates a different screen or action the user can navigate to. There can only be one selected tab, and the currently selected tab is indicated by the color change and bar beneath the icon. From left to right the screens associated with each tab are: Listings, Search, Edit Listing, and Profile.

Pressing the home icon tab will take the user to the Listings screen for their current community. Pressing the search icon tab takes the user to the Search screen. Pressing the sales tag icon tab will pop up a photo-taking screen, and upon taking a photo takes the user to the Edit Listing screen. However, if they are not logged in and in guest mode, they are presented the Login screen. Pressing the user icon tab takes the user to the Profile screen. However, if they are in guest mode, they are given the option to either login or to change their community.

As part of the iOS Human Interface Guidelines the tab bar is translucent. This can be seen below. The translucency effect is a result of colors from the user interface beneath the tab bar partially showing through to give the user contextual awareness and depth of where they are in the app. This specific translucency effect was generated from the Listings beneath the tab bar, and can be seen along with the rest of the user interface in the Listings figure.

***Figure 3**. Tab bar navigation*

Listings

The Listings screen is the main screen of the app. It is navigated to by selecting the first tab bar tab from the left. From here, a tiled layout of all listings posted in the user's current community is displayed. Once a listing has gone unclaimed for 2 weeks it can no longer be seen on this screen. For each listing tile the image of the listing is shown, along with a price in the bottom right of the tile. Touching a tile selects it and brings the user to the Listing Details screen. To make it clear to the user which community they are currently in, a label is shown at the top left with the name of the current community. To the right of that label is the Filter button. Pressing the filter button animates the Listings Filter screen into appearance. When the user wishes to refresh the listings shown, they can "pull to refresh."

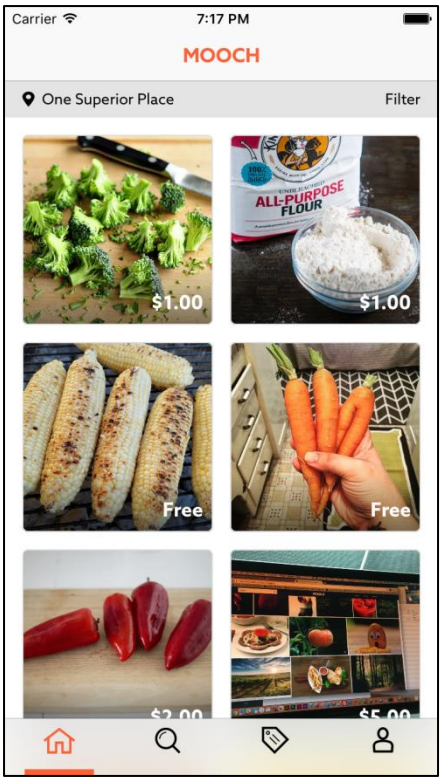


***Figure 4**. Listings*

Listings Filter

The Listings Filter screen provides the user with options for filtering and sorting a group of Listings. There are four different filtering options the user is provided with: sort ordering, food category, posted within duration, and price range. Pressing the sort ordering, food category, posted within duration rows shows a screen with the associated options. Pressing the Clear button in the upper left will remove all filters, and return the user to the Listings screen. Pressing the Done button in the upper right will apply all selected filters, and return the user to the Listings screen.

For sort ordering there are four options: best match, lowest price, highest price, and newest. The best match option is essentially a placeholder that takes the food category, posted within duration, and price range the user selects and returns all listings that match those filters in no particular order. The lowest price and highest price options take the food category, posted within duration, and price range the user selects and then sorts them based on price, respectively. The newest option takes the food category, posted within duration, and price range the user selects and returns all listings matching those filters, sorted by the most recently posted listings. By default, the selected sort ordering option is Best Match.

The food category option allows the user to choose a single food category to filter by. These food categories are predefined.

The posted within duration option allows the user to filter out listings that were posted beyond a specific amount of time. Users can choose to only show listings posted within: 24 hours, 2 days, 5 days, 7 days, and 2 weeks. By default, the selected posted within option is 2 weeks.

The price range allows users to choose the minimum and maximum allowable price ranges. The lowest price that can be selected is $0, and the highest price that can be selected is $200. The user can drag the double-ended slider to select this range.

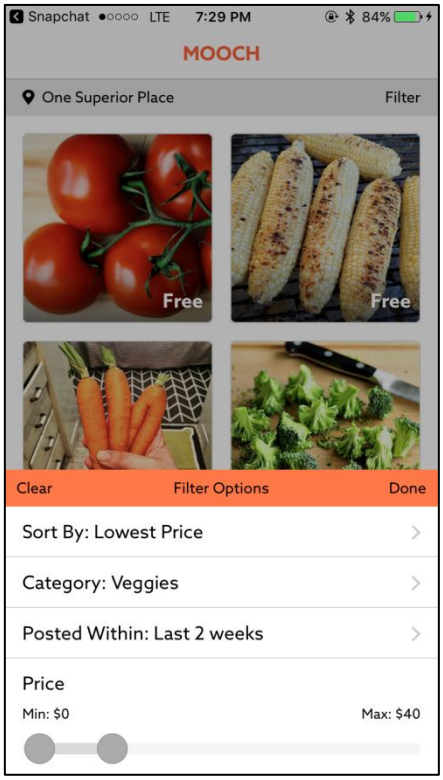


***Figure 5a**. Listings filter*

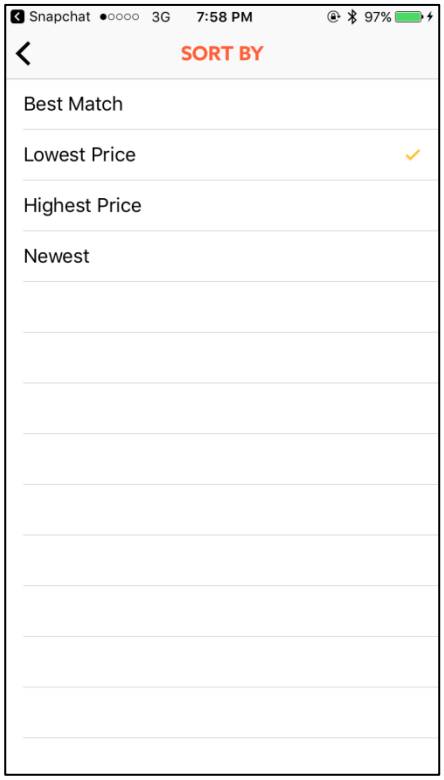


***Figure 5b**. Listings filter, sort by options*

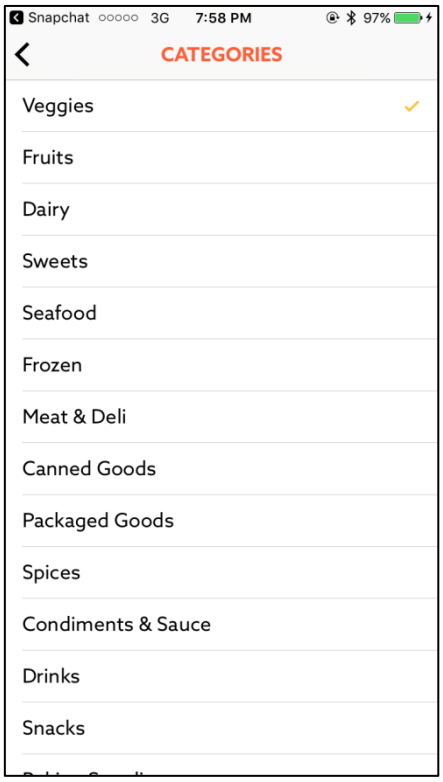


***Figure 5c**. Listings filter, food category options*

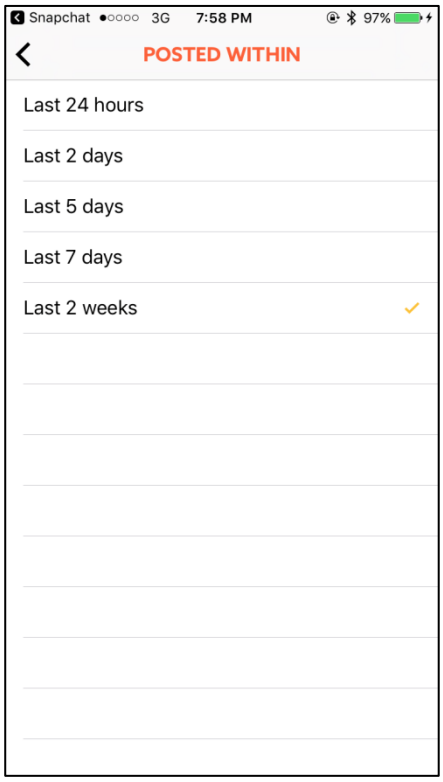


***Figure 5d**. Listings filter, posted within options*

Listing Details

The listing detail screen shows the information for a listing. The listing information shown is the listing's image, title, date posted, quantity, price, and details. The screen is scrollable such that the user can "pull down" on the listing image to make the image grow to the full size of the screen and be fully visible. Depending on the context the screen is navigated from, there are 4 different major configurations for the content that shows beneath the listing information:

1. Viewing other user's available listing
2. Viewing other user's accepted listing
3. Viewing current user's available listing
4. Viewing current user's accepted listing

Viewing other user's available listing

This Listing Details configuration can be reached from the Listings screen, or from the Contact History portion of the Profile screen if the seller hasn't accepted the exchange request. Below the listing information is the Contact Seller button, View Seller Profile button, listing description, and About Seller section. Pressing the Contact Seller button creates an exchange request between the user and the seller for this listing. Once selected, the button changes to a dimmer color and becomes disabled. The View Seller Profile button shows the Seller Profile screen. The About Seller section shows the seller's profile photo and name, and a label informing the user how a listing exchange can be completed to reveal the seller's contact information.

Viewing other user's accepted listing

This Listing Details configuration can be reached from the Contact History portion of the Profile screen when the seller has accepted the exchange request. There is a banner saying "LISTING ENDED" to indicate that the listing is no longer available, and the seller has accepted the exchange for the current user. Below the listing information is the View Seller Profile button, listing description, and About Seller section. The View Seller Profile button shows the Seller Profile screen. The About Seller section shows the seller's profile photo, name, email, phone, and address. Pressing the seller's email opens up the Apple Mail app with the sender field of the email prepopulated. Pressing the seller's phone number presents the user with the option to call or text the user. If the user selects the call option, they are taken to the Apple Phone app. If the user selects the text option, they are taken to the Apple Messages app.

Viewing current user's available listing

This Listing Details configuration can be reached from the My Listings portion of the Profile screen if the current user hasn't received or accepted any exchange requests. Below the listing information is the End Listing button, listing description, and Interested Buyers section. Pressing the End Listing button will forever delete the listing. The Interested Buyers section shows the users that have started an exchange with the current user for this listing, if there are any. Each interested buyer row shows the buyer's profile picture and name, and has an Accept button. Pressing the Accept button for a prospective buyer will complete the exchange with that buyer for the listing. This means the listing is no longer visible, and is in an accepted state. If there are no interested buyers, a label

saying “Currently No Interested Buyers” informs the user. The listing can also be edited by pressing the settings gear icon in the top right.

Viewing current user’s accepted listing

This Listing Details configuration can be reached from the My Listings portion of the Profile screen if the current user has accepted an exchange request for the listing. There is a banner saying “LISTING SOLD” to indicate that the listing is no longer available, and the current user has accepted the exchange for the shown buyer. Below the listing information is the listing description and About Buyer section. The About Buyer section shows the buyer’s profile photo, name, email, phone, and address. Pressing the buyer’s email opens up the Apple Mail app with the sender field of the email prepopulated. Pressing the buyer’s phone number presents the user with the option to call or text the buyer. If the user selects the call option, they are taken to the Apple Phone app. If the user selects the text option, they are taken to the Apple Messages app.

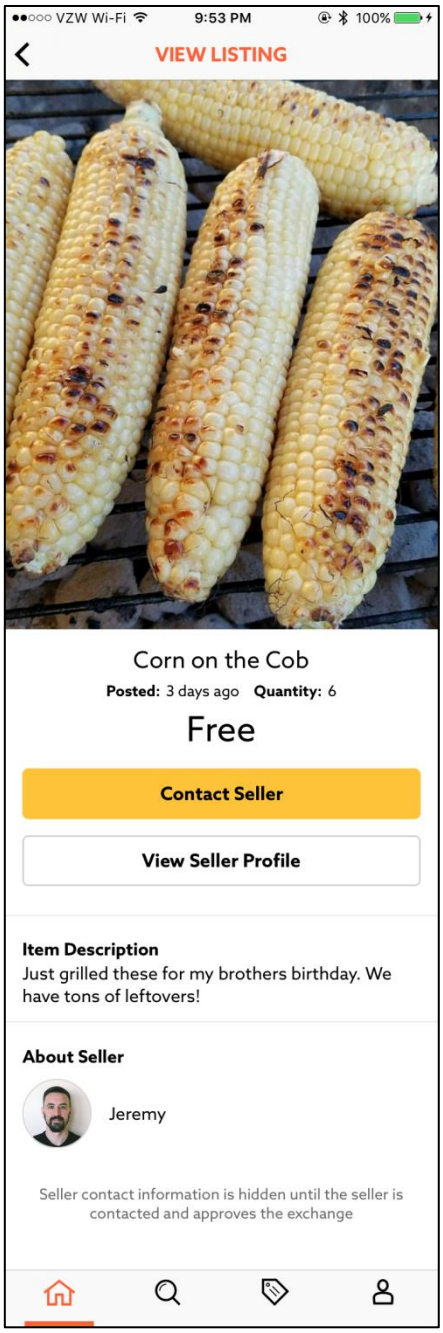


***Figure 6a**. Listing details, other user's available listing*

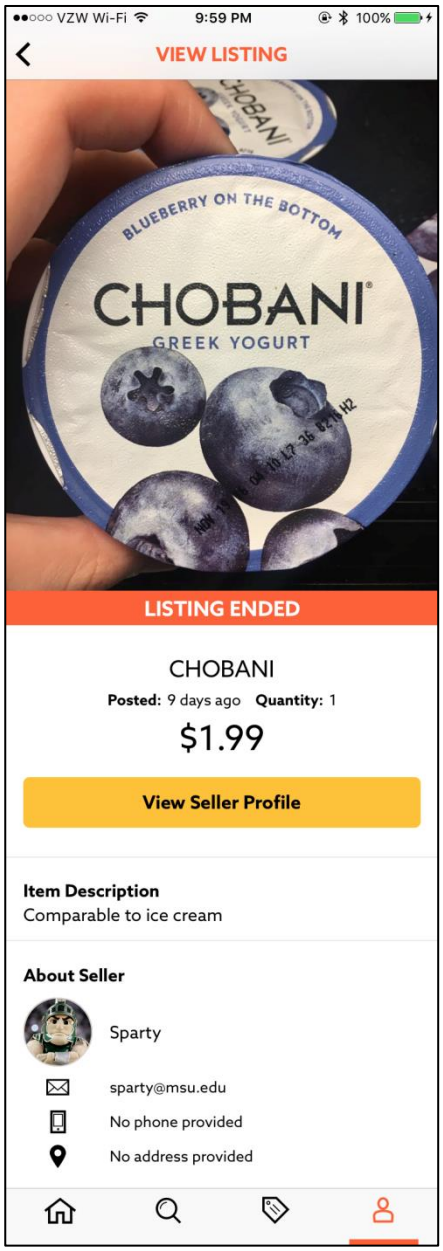


***Figure 6b**. Other user's accepted listing*

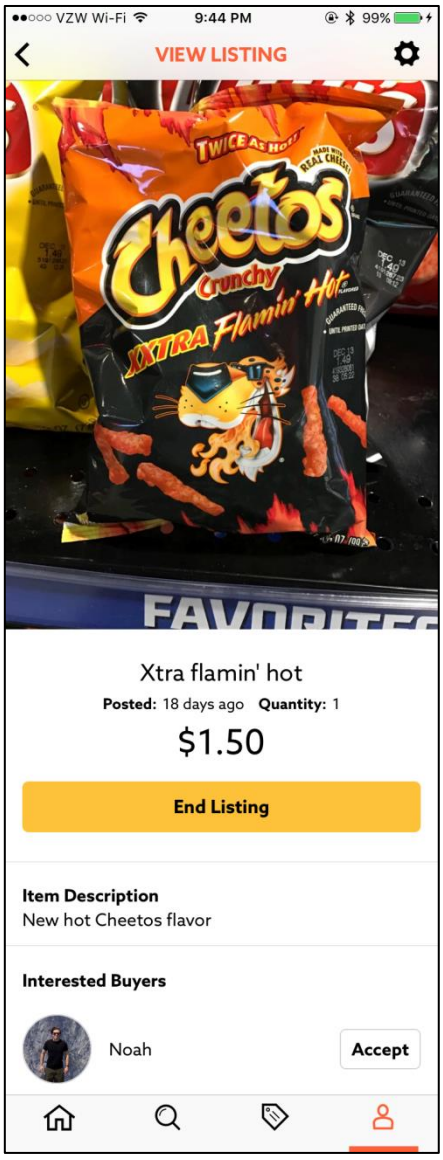


***Figure 6c**. Listing details, current user's available listing*

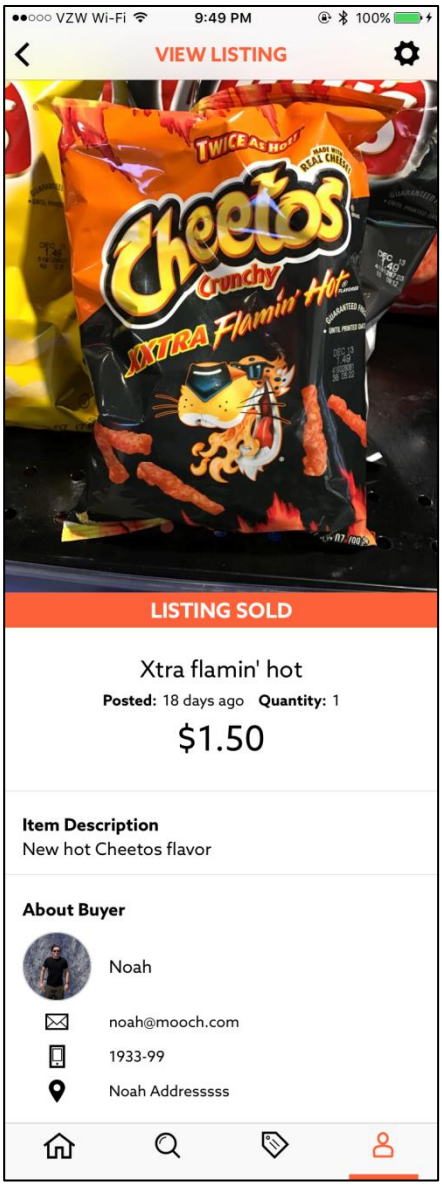


***Figure 6d**. Listing details, current user's accepted listing*

Seller Profile

The Seller Profile screen shows the user information of a seller. It can be reached from a Listing Details screen. The seller's profile photo, name, and current community are at the top of the screen. The bottom half shows other listings the seller has posted. They can be selected, which will show the Listing Details screen for them.

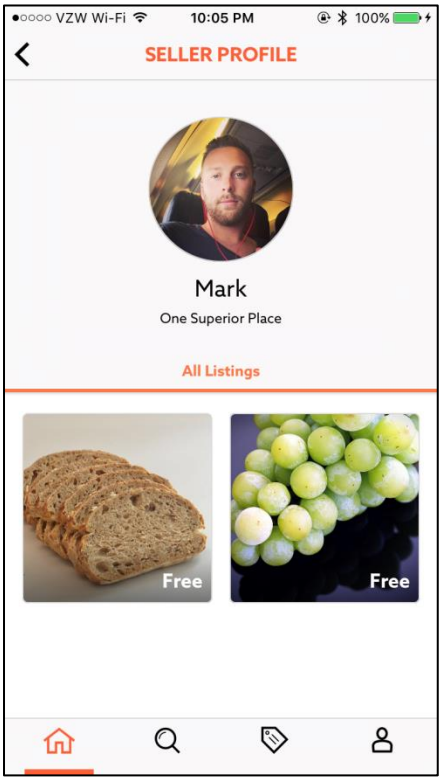


***Figure 7**. Seller profile*

Search

The Search screen allows the user to search through listings in their current community. A user can type in the navigation bar with a search field at the top of the screen, and only listings whose name or description match the search will be shown. As the user types the matching listings are updated. These listings appear under the navigation bar. The user can also select a specific food category, and then only listings in the selected food category will be shown. The listings in the selected food category can then also be searched in the same way the listings could be searched before a category was chosen. If the user no longer wishes to only search listings in the selected food category, they can press the back button at the top left of the screen.

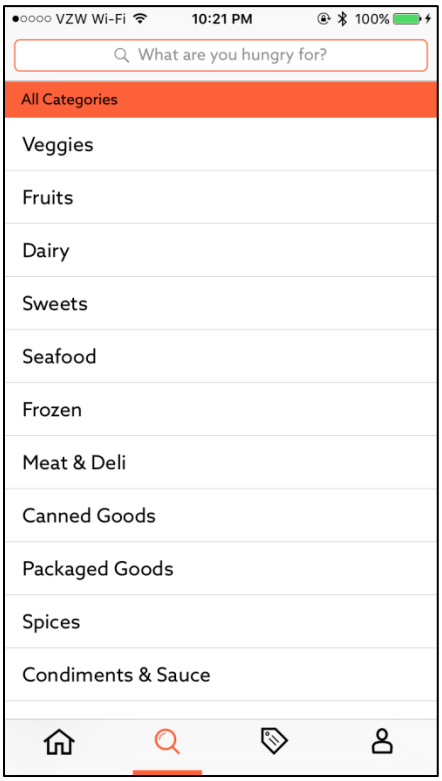


***Figure 8**. Search*

Edit Listing

The Edit Listing screen is where the user creates or edits a listing. When creating a listing this screen appears after taking a picture of the item by pressing the sales tag icon tab in the tab bar. When editing a listing the screen is reached from the Listing Details screen of a listing for the current user. The image cannot be changed once on this screen. The user can change the title, price, quantity, description, and category. All the preceding fields are required to post other than the description field. Once they are finished, they press the Post button. The fields are all checked that they are filled and valid; otherwise an alert appears informing the user how to fix the problem. Upon successfully posting the screen is dismissed and either creates the new listing or finishes editing the existing one. The user can also press the Cancel button to leave the screen.

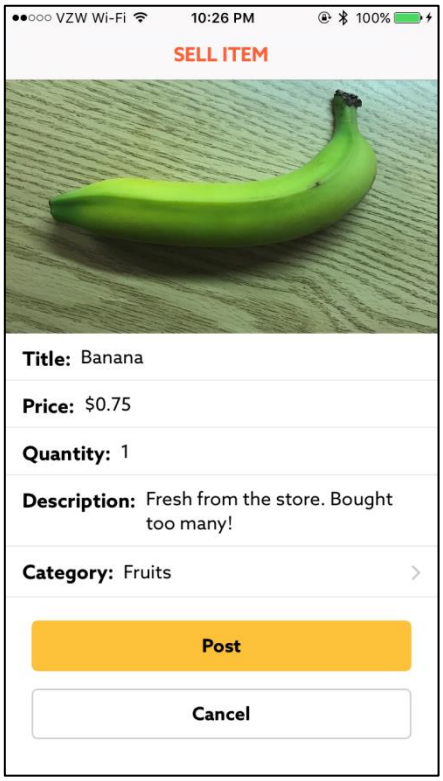


***Figure 9a**. Edit listing, creating listing*

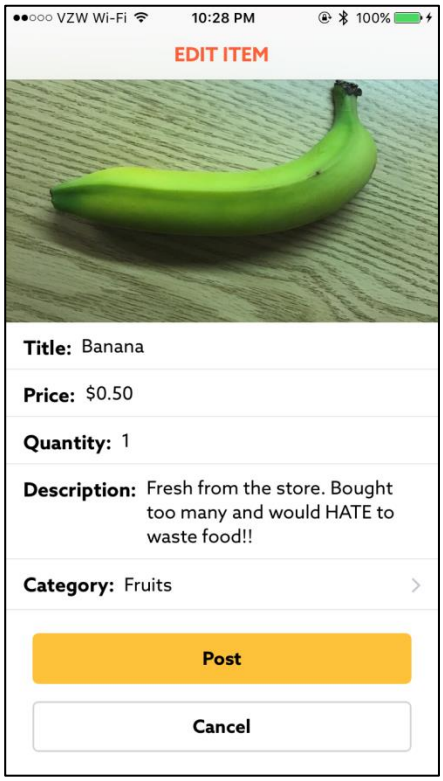


***Figure 9b**. Edit listing, editing existing listing*

Login

The Login screen is where the user signs into Mooch.. The username and password text fields are in the middle of the screen. The Sign In button is below the text fields, and the Create Account button is below the Sign In button. The user types their email and password into the respective fields. The email is validated by a regex, and the password is checked that it is at least 6 characters. Once both those conditions are met, the Sign In button becomes enabled. Upon successfully signing in after pressing the Sign In button, the user is returned to the screen they came from. When the Create Account button is pressed the user is taken to the Edit Profile screen. Pressing the X button at the top left of the screen will also return the user to the screen they came from.

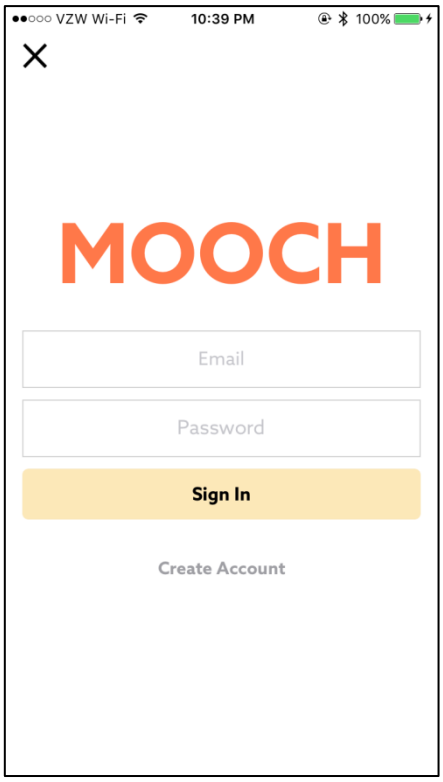


***Figure 10a**. Login, fields empty*

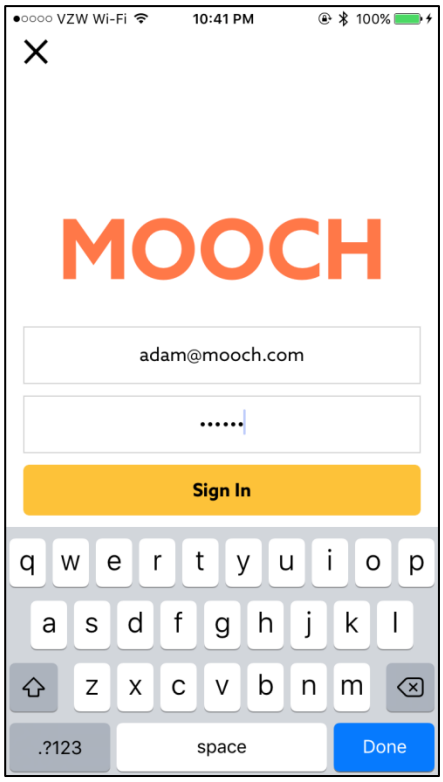


***Figure 10b**. Login, fields filled and valid*

Profile

The Profile screen shows a user's information and allows them to change their information. A user can see their profile photo, name, and current community name. There are two tabs in the middle of the screen, My Listings and Contact History. One of these tabs is selected at a time, and the selected tab is indicated with bold text and a bar below it. A tab is selected by pressing the text, like a button. Below the tabs the applicable listings show and can be scrolled through. The My Listings tab shows the listings the current user has posted, and the Contact History tab shows the listings of other users the current user has contacted. In the top right of the screen is a settings gear button. Pressing that button gives the user the following options: Edit Profile, Change Community, and Log Out. The Edit Profile option takes the user to the Edit Profile screen. The Change Community option takes the user to the Community Picker screen, where they can change their community. The Log Out button signs out the user and puts the app into guest mode.

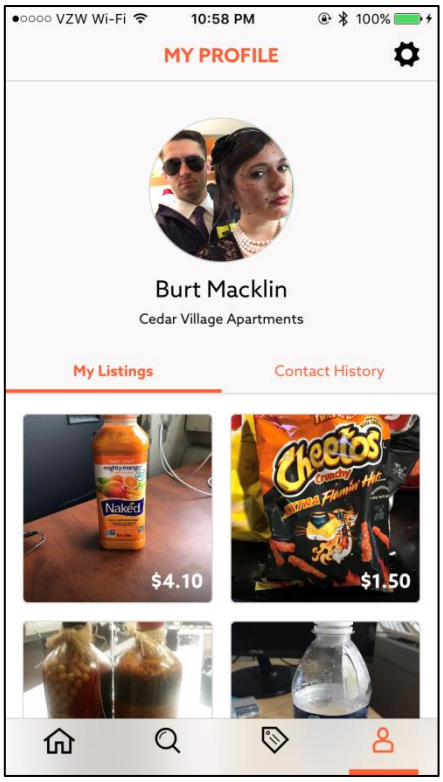


***Figure 11a**. Profile, "My Listings" selected*

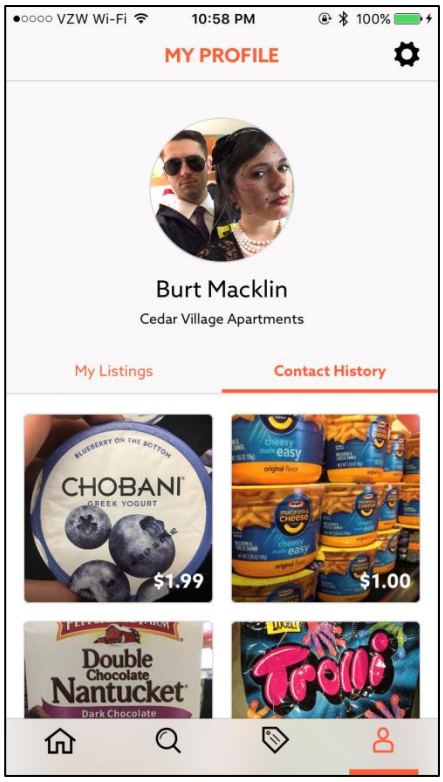


***Figure 11b**. Profile, "Contact History" selected*

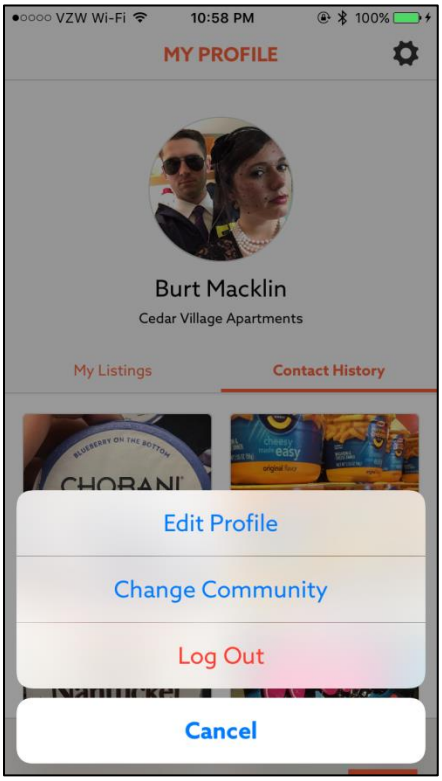


***Figure 11c**. Profile, settings gear button pressed*

Edit Profile

The Edit Profile screen is where a user can create a new account or change their existing information. When creating an account this screen is reached from the Login screen. When editing a profile the screen is reached from the Profile screen of the current user. When creating an account a user can fill out their photo, name, email, phone number, address, password, and community. When there isn't already a photo picked, the photo field is filled by pressing the circular icon at the top. When a photo is already selected it is changed by pressing the edit icon at the bottom right of that field. All the preceding fields are required for a profile other than the photo and address fields. When editing an account a user can only change their photo, name, email, phone number, and address. Once they are finished, they press the Complete button. The fields are all checked that they are filled and valid; otherwise an alert appears informing the user how to fix the problem. Upon successfully completing the screen is dismissed and either creates the new account or finishes editing the existing one. The user can also press the Cancel button to leave the screen.

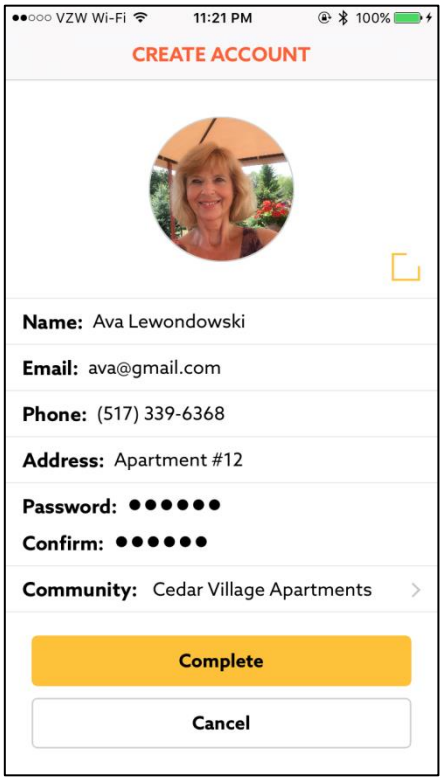


***Figure 12a**. Edit profile, creating account*

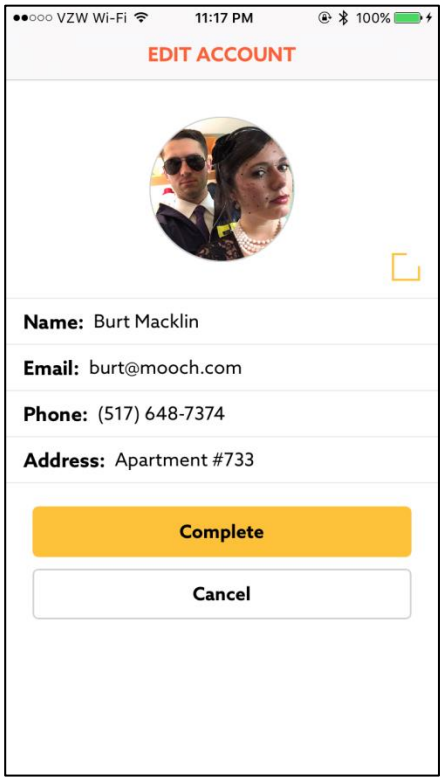


***Figure 12b**. Edit profile, editing account*

Use Cases

Mobile Application Use Case for Food Suppliers

Ryan lives in a large apartment building in New York City. He opens up his refrigerator, and sees that he has two new jugs of milk. He checks the expiration dates and realizes he will only be able to get through one jug of milk by the time they expire. Ryan doesn't like the idea of wasting food, and he also likes making easy money. Ryan remembers a friend mentioning a food-sharing app called Mooch, and downloads the app from the app store.

Ryan opens the app. Mooch allows people to use it without signing in initially, but they still have to pick their community. Ryan sees his apartment building and selects it. He is taken to the Listings screen, which shows all the food items people have added in his community. He sees the button to add a food listing, and presses it. Since he doesn't have an account yet, he is taken through the process of making one. He is then directed to a screen to create a new listing. Listings need pictures, so Ryan takes a photo of his milk jug. He fills in the title, "Fresh 2% milk jug," and sets the category to "Dairy." He sees that the price has defaulted to free, but instead decides to charge $1.50 for it. Now that all the required information is filled out, he finishes creating the listing, and now other Mooch users can see it in the Listings screen. Ryan closes the app.

Five minutes later, he gets a push notification that someone wants his listing. Ryan clicks on it, the Mooch app opens, and Ryan is taken to the Listing Details screen for his milk listing. He sees Phil as an interested buyer and accepts the exchange request. Ryan and Phil can now see each other's contact information. Phil texts Ryan and they meet. After some small talk they realize they have a friend in common. They exchange the milk and money, and decide they should hang out that weekend.

Mobile Application Use Case for Food Demanders

Tanay lives in a large apartment in downtown Chicago. While at home, he decides he would like to cook his favorite meal, dragon noodles. He checks the recipe and sees he has everything but the tablespoon of brown sugar it needs. Tanay doesn't want to make a trip to the grocery store just to buy a bag of brown sugar when he only needs a tablespoon. Like most people in the area he doesn't own a car and the nearest grocery store is a few blocks away. So, Tanay opens the Mooch app.

He has already signed into the app before and is taken straight to the Listings screen. He doesn't see brown sugar right away, so he searches the listings in his community by text, and finds a brown sugar listing from a few days ago posted by Susie. He presses the button to claim the listing and waits for Susie to respond. To pass the time he goes back to Listings screen and scrolls through the listings of food, enticed by the pictures. He sees a can of Vanilla Coke. Tanay likes Vanilla Coke, but not enough to bother carrying a 12-pack of it home. It turns out seeing a Coke makes him crave one, so he impulse claims the can of Coke, too. Soon after an alert pops up, notifying him that Susie has accepted his claim request. Now that they have completed the exchange, Tanay can see Susie's contact information. Tanay emails him, and they successfully complete the exchange in the lobby of their building.

Web Application Use Case for Mooch Administrators

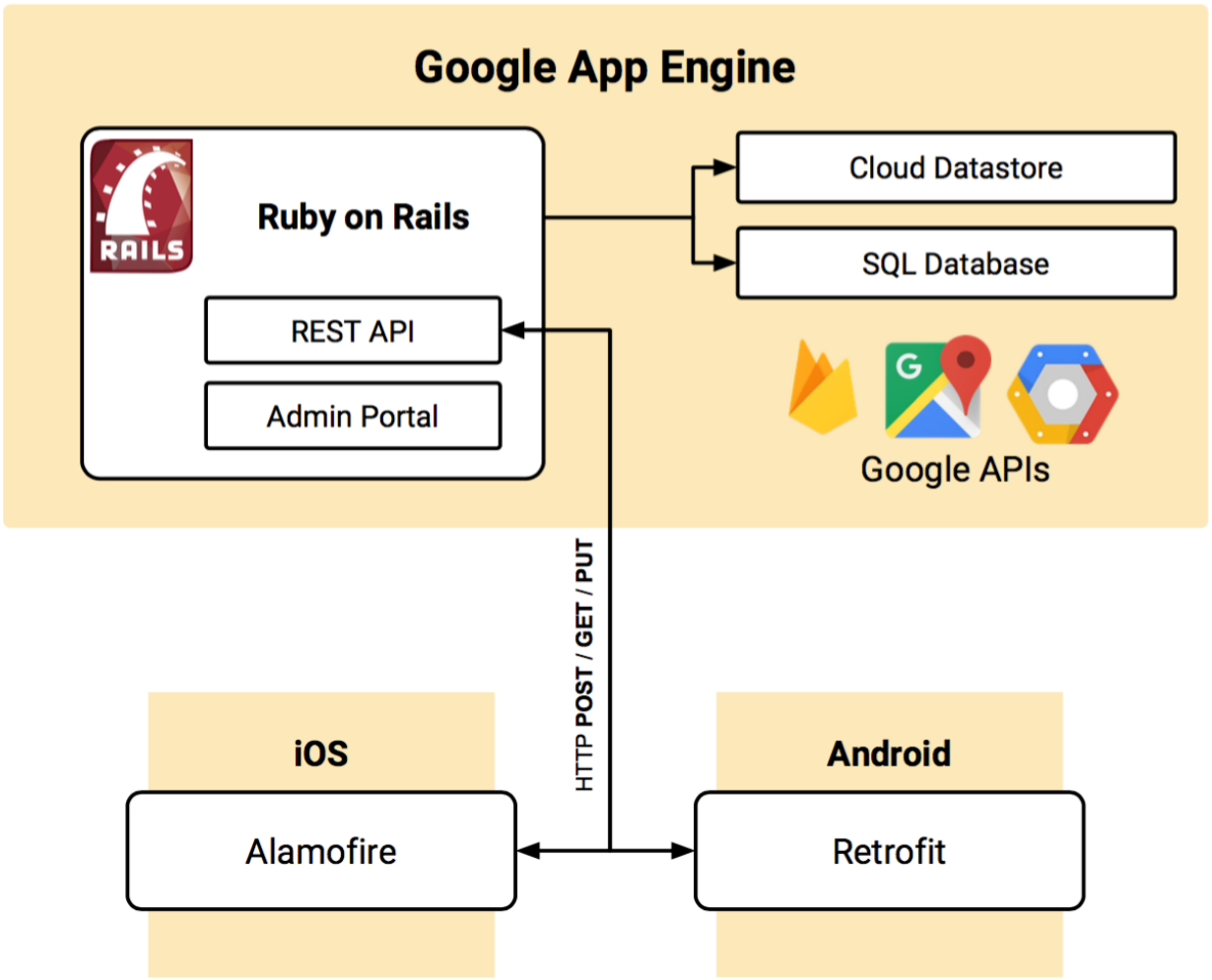
Ian is an administrative user of Mooch. He has a new community to add to the system. Ian navigates to the URL for the web app. He enters his username and password, and is logged in. He clicks the Communities button and is directed to the page for managing communities. He clicks the Add Community button, fills out the required information, and clicks Save. The community is added to the database and all Mooch users will be able to select it now.

Ian is also curious how user growth and engagement has been going. He navigates back to the home page. From there, he sees how many users there are, how many listings have been posted, how many listings have been successfully completed, the most common listing categories, and other useful information.

Technical Specifications

This section details the various technologies that power the Mooch app and its RESTful Application Program Interface (API). Specific implementation of the system architecture, database schema and network flow can be found in the figures below.

System Architecture



The Mooch API is hosted on Google App Engine: a scalable platform as a service used to host web apps. The databases are hosted on App Engine, which utilizes both SQL and NoSQL databases. Our SQL Database is used to store model data and uses MySQL2, which integrates nicely with Ruby on Rails. Our NoSQL Database is hosted on Google Cloud Datastore and is used to host our BLOB (Binary Large OBject) data, including Listing and User pictures.

API

The Mooch API was written using test-driven development with RSpec: a behavior-driven testing framework for Ruby. The API handles creation, modification, and deletion of the User, Listing and

Transaction models. The API routes follow the standard REST URL scheme: website.com/api/:model/:id. A GET request is used to request data, PUT is used to modify existing data and POST is used to create new data. With these three HTTP methods, the Mooch app could function perfectly fine, but the app wouldn't be very secure if the apps can create, edit and delete anything they wanted. To secure the data for each user, we've added a login route, which returns an authentication token that is used in protected routes, such as creating, editing, and deleting a listing.

Google Cloud Engine provides direct access to a handful of APIs we've taken advantage of to improve the Mooch experience.

First, we used the Google Maps API in our Admin portal to make adding and editing Communities a breeze. When creating a community, a Mooch admin can use the embedded Google Maps view to set the location marker on the exact location of the Community. The marker's latitude/longitude value is stored in the Community model and is used when a Mooch user opens the Community Picker.

Second, we used the Google Vision API to determine the dominant color in each listing image. This color serves as a temporary background color for each cell in the listings view, which greatly improves the user experience.

Third, we used Google's Firebase Cloud Messaging to send push notifications to Android and iOS devices.

Mobile Apps

The iOS and Android mobile apps communicate directly with the API by making HTTP requests at the Mooch API URL (http://mooch-test.appspot.com/:model_here). On launch, the mobile apps send a GET request to the /listings URL, which gets the most recent listings in JSON format. Using this data, the mobile apps create their app models and display them in the main listings view of the app.

The listing images are crucial to the look and feel of the app. As such, a lot of time has been put into optimizing the storage and retrieval of these images. We decided to send image URLs instead of embedded URI's because it allows the mobile apps to asynchronously download the images after the main view has been laid out.

Push notifications alert Mooch users when they receive an exchange request for a listing or if their exchange has been accepted. The API sends push notifications to devices using device-specific tokens called destination tokens. Each User object has a device_token value, which is used to send automated push notifications when an Exchange is created or accepted. Using the Admin Portal, a Mooch admin can also send a custom message to all Mooch users.

Web Admin Portal

Admin users have access to the web admin portal: an interactive web app that has full control of the data in the app. With the admin portal, admin users can view, edit, and delete Users, Listings, Exchanges, Communities, Categories, and Push Notifications.

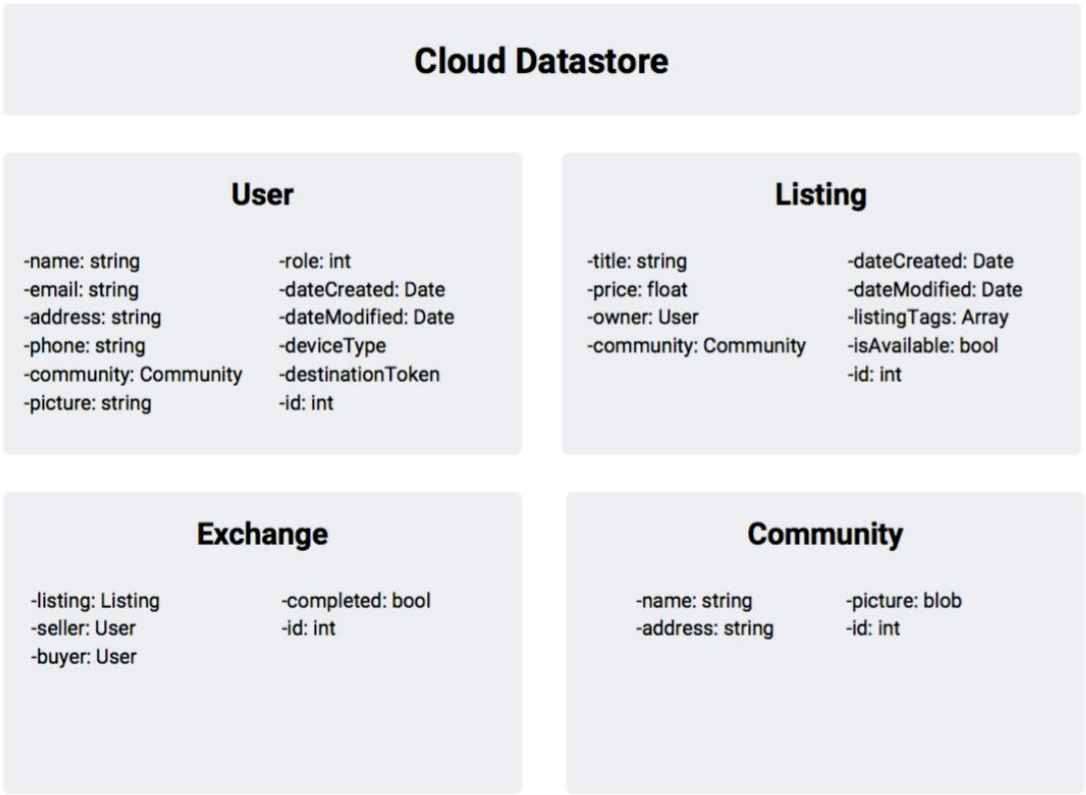
The main page of the admin portal is the Dashboard, which provides live data analytics in the form of both tables and interactive javascript charts. Using AJAX and jQuery, the admin portal communicates

with the API to pull the latest data every 5 seconds. If the data has changed, then the information is re-drawn on the screen. A great example is when a user adds a listing and the Listing Ticker increments live.

Software Technologies

- Ruby on Rails
- Google App Engine
- MySQL
- Google Cloud Datastore
- Javascript, jQuery, AJAX
- HTML5 & CSS3
- Swift 3 (iOS versions 9 and 10)
- Android (Java, API 15+ Ice Cream Sandwich)

Database Schema



Development Environments

The REST API was written in Ruby with a Macbook Pro using the standard Ruby on Rails local server. The iOS app was written using Swift 3 on a Macbook Pro with Xcode. The Android app was written on an iMac and Windows PC using Android Studio. The admin web portal was written using Javascript, jQuery, and AJAX on a Macbook Pro using the standard Ruby on Rails local server.

Risks

Android Application Fundamentals

- No previous experience with making an Android application
- Mitigation: Researching Android programming through online tutorials and documentation and some MSU coursework
- Status: Used all of the mentioned resources to create the working Mooch application.

Completing Transactions in Mooch

- Need to figure out how to make transactions reliable and secure. Users will need to be able to contact each other but without revealing more personal information than is comfortable. The app must also be able to handle potential network errors or loss of network connection.
- Mitigation: iOS app checks for a valid network connection. Brainstormed with Whirlpool and design team for ways that users can complete transactions, such as push notification alerts for sale approval and template messages for buyers to send sellers (text or email).
- Status: Both apps handle possible network errors and display notices to the user that an error has occurred. Push notifications are used, but if those do not come through, the listing details screen will have all the necessary information to complete a transaction.

Security of Users

- Users' contact information needs to be securely shared within the application to avoid unauthorized distribution to third party sources.
- Mitigation: A seller's contact information will only be shared with a buyer once the seller has confirmed the transaction.
- Status: The seller's contact information is now only available to a buyer from the listing details screen after that buyer has been approved by the seller.

Database Usage in Application

- Need databases of food types and community locations. Possible food types will be a very extensive list.
- Mitigation: Create a small set of database entries for communities initially. Search for an existing, publicly available database for the types of food items.
- Status: Decided against using "food types" for tags, and instead have users sort listings into preset categories, similar to food sorted by isle in a grocery store. We have manually created several communities for real apartment complexes.

Schedule

Week 1 (8/31 – 9/2)

- Team and client established, team roles determined
- Conference call with Whirlpool to get started on the project
- Began brainstorming initial models for API

Week 2 (9/6 – 9/9)

- Setup GitHub repositories for projects
- Basic work on setting up the API
- Conference call to clarify features that will be included

Week 3 (9/12 – 9/16)

- Status Report presentations (9/14)
- Introduced to design team from JohnsonRauhoff, working with them via Basecamp
- Set up weekly Hangout meetings
- First draft of user app flow diagram from JohnsonRauhoff

Week 4 (9/19 – 9/23)

- Project Plan presentations
- Finalize app flow design with JohnsonRauhoff/Whirlpool
- Continue building API after more design information is received
- Database/tables set up to reflect the API models, have some dummy data to work with

Week 5 (9/26 – 9/30)

- iOS and Android applications have appropriate libraries hooked up to make API calls
- Final rapid wireframes from JohnsonRauhoff received

Week 6 (10/3 – 10/7)

- iOS and Android applications can create, modify, and view listings via API calls
- First set of finalized wireframes from JohnsonRauhoff received

Week 7 (10/10 – 10/14)

- Alpha version of application finished
- Both apps support user creation, logging in

Week 8 (10/17 – 10/21)

- Alpha presentations
- Received all final wireframes from JohnsonRauhoff
- Used wireframes to update existing UIs and keep building the apps
- Started to figure out the steps of user exchanges

Week 9 (10/24 – 10/28)

- First finalized mock ups from JohnsonRauhoff received
- Adjusted any existing UI features
- Steps for exchanges between users functioning

Week 10 (10/31 – 11/4)

- All final mock ups received from the JohnsonRauhoff team
- Fixed UI inconsistencies and missing features
- Figured out push notifications for exchanges

Week 11 (11/7 – 11/11)

- Both apps feature complete
- Added in final design elements

Week 12 (11/14 – 11/18)

- More extensive user testing to test for bugs
- Held an informal "launch party" to get help with finding bugs and populating the app with data
- Started project video

Week 13 (11/21 – 11/25)

- Continued to work on project video
- Fixed bugs that were found through user testing

Week 14 (11/28 – 12/2)

- Finalized the project video

Week 15 (12/5 – 12/9)

- 12/5: Project videos
- 12/7: All project deliverables due
- 12/8: Design Day setup
- 12/9: Design Day