

# PICKERpals

## Design Document

Matthew DeGenaro, Manoj George, Sean Spencer,  
Gianluca Solari, Joseph Whittier, Ryan Beebe,  
Joseph Mcilvaine, Joseph Antaki

Team Leader: Joseph Mcilvaine

<https://github.com/RyanBeebe1/SeniorProjectPICKERpals>

Slack: JoeCubed.slack.com

## Table of Contents

<b>High Level Description</b>	<b>3</b>
<b>Problem Solving Approaches</b>	<b>4</b>
<b>Design Mockups - Home Screen</b>	<b>5</b>
<b>Screen Navigation</b>	<b>10</b>
<b>Flow Diagrams</b>	<b>11</b>
<b>Technology Stack</b>	<b>13</b>
<b>Backend Information</b>	<b>14</b>
<b>Authentication and Security</b>	<b>15</b>
<b>Input and Output</b>	<b>16</b>
<b>Database Schema</b>	<b>17</b>
<b>Restful Endpoints</b>	<b>18</b>
<b>Team Responsibilities</b>	<b>20</b>
<b>Mid-Assessment Goals</b>	<b>21</b>

## High Level Description

The purpose of PickerPals is to help facilitate a fast and simple transaction between two groups of people; the person getting rid of their unwanted items, and the person searching for things being thrown away.

When the user opens PickerPals, they will be greeted with a feed page that displays listings containing nearby items for which people are throwing away. This listing will feature important information regarding the items, and will allow for communication between both parties involved. For a user to post new items to be 'picked', they will have to create an account by logging in through google.

The chat screen will start by displaying all previous people whom with the user has previously communicated. A user image and their name will be displayed, as well as the last message. Once a user is selected, the chat screen will be displayed, allowing a basic back and forth communication between two users.

Back on the feed screen, there will be a button in the lower right hand corner that allows users to add a listing for a new item. This will bring the user to another screen wherein they fill out the required fields for adding a new item, which will be added to the feed. A user can view the items they have added on their profile screen, as well as their own user information.

## Problem Solving Approaches

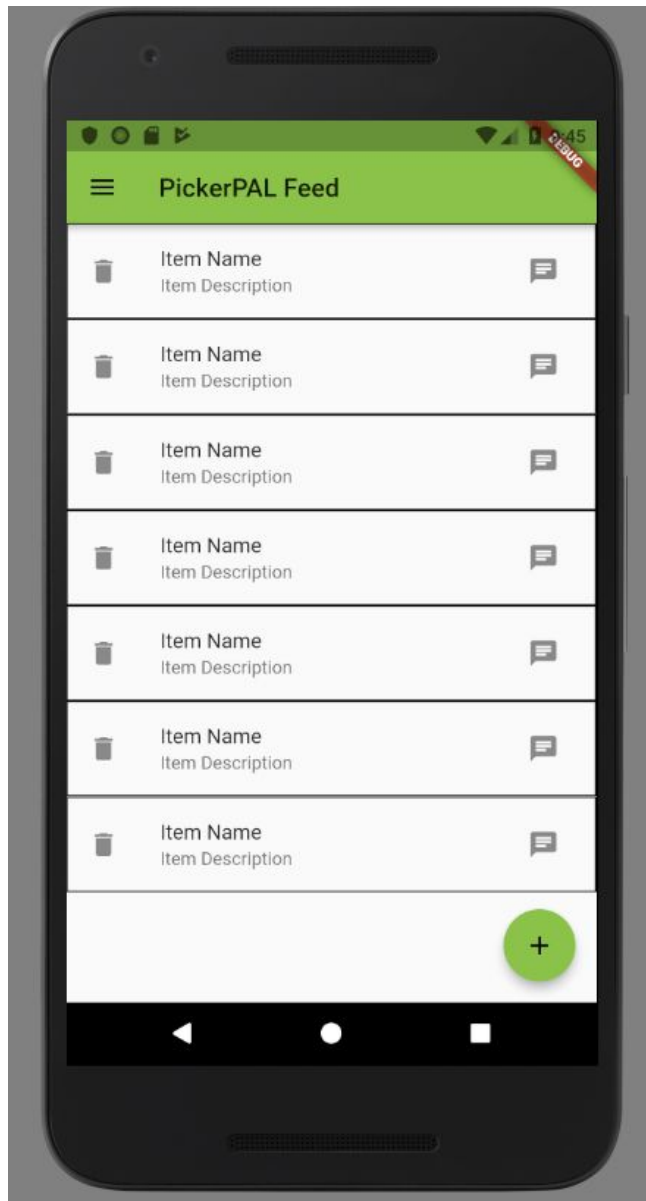
Originally, we considered using Google Firebase for the backend database since we are also using it for Google OAuth. After the class lecture about Amazon's endpoint services, we decided that interacting with these endpoints via Python with a Flask microframework was a lighter-weight way to maintain a database, and that it would be more productive since much of the team has familiarity with Python.

As for our web framework, we considered both Django and Flask, but ultimately chose Flask because it fit better for our purposes since it is considered much more bare-bones, keeping with the pattern of lightweight choices as we made for Amazon endpoints.

One of the problems that might be faced throughout development is the fetching/displaying of listing information on the front-end. For the dynamic feed to be viable, it needs to load only 20 listings at a time. Otherwise, we would end up with a bunch of cached data on the front-end that may never be viewed. So, we need to ensure that when querying the database, we are receiving listings in the correct order and not creating any duplicates. This will require configuring and caching of data on the back-end.

## Design Mockups - Home Screen

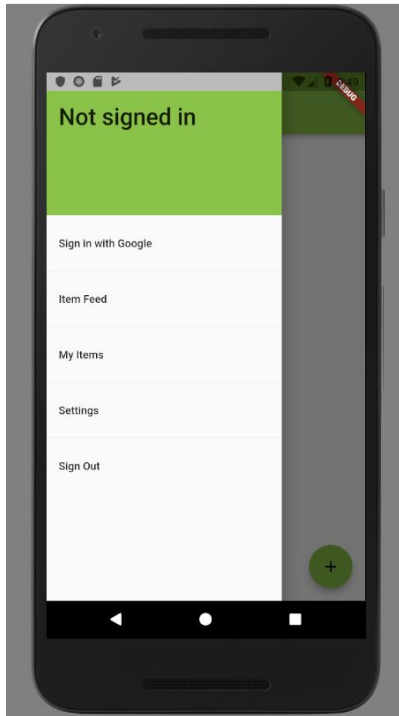
This is the first screen that users will be greeted with when they open the map. This is just conceptual at the time, so it contains no real data. From here, users will be able to navigate to any other screen. None of these designs are set in stone.



## Design Mockups - Drawer

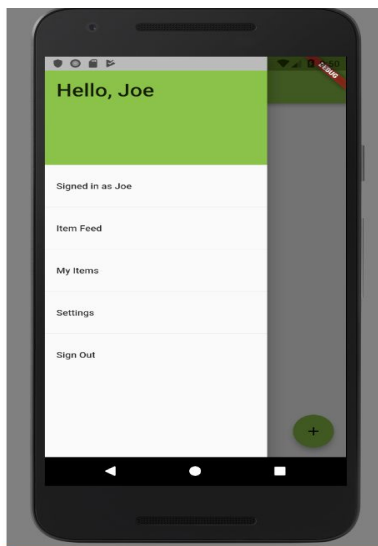
### Not Logged in -

This is how the screen will appear when a user is not signed in.



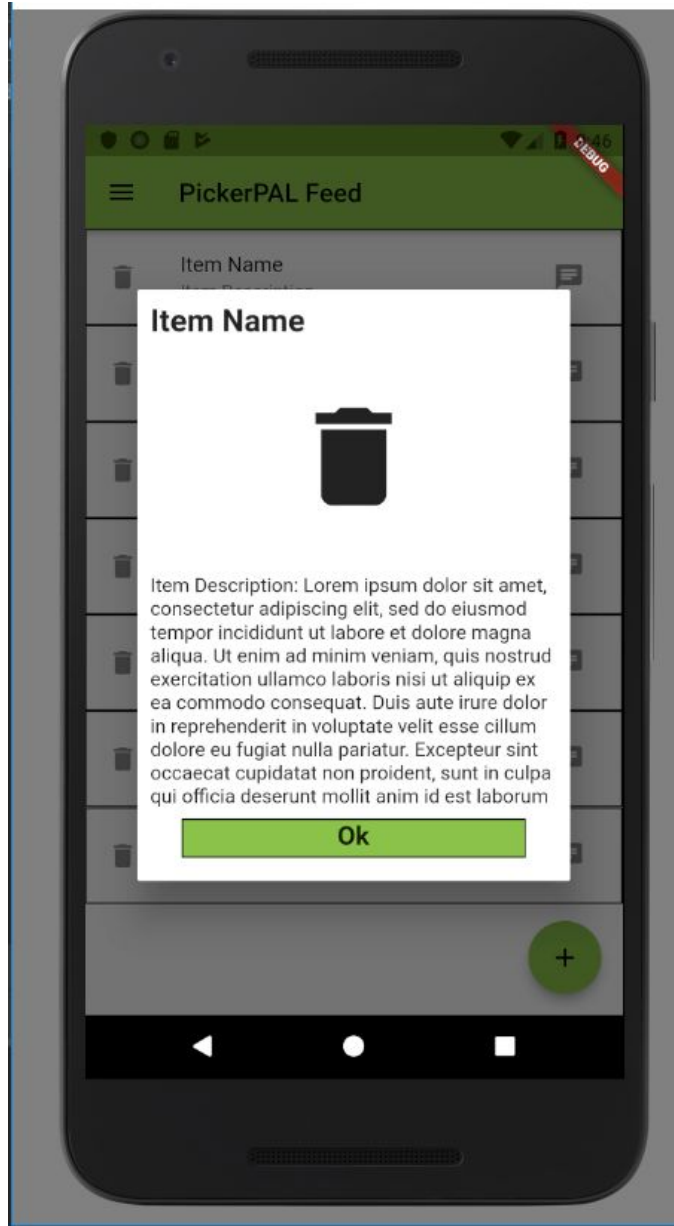
### User Logged in -

This is how the screen will appear when a user is logged in through Google.



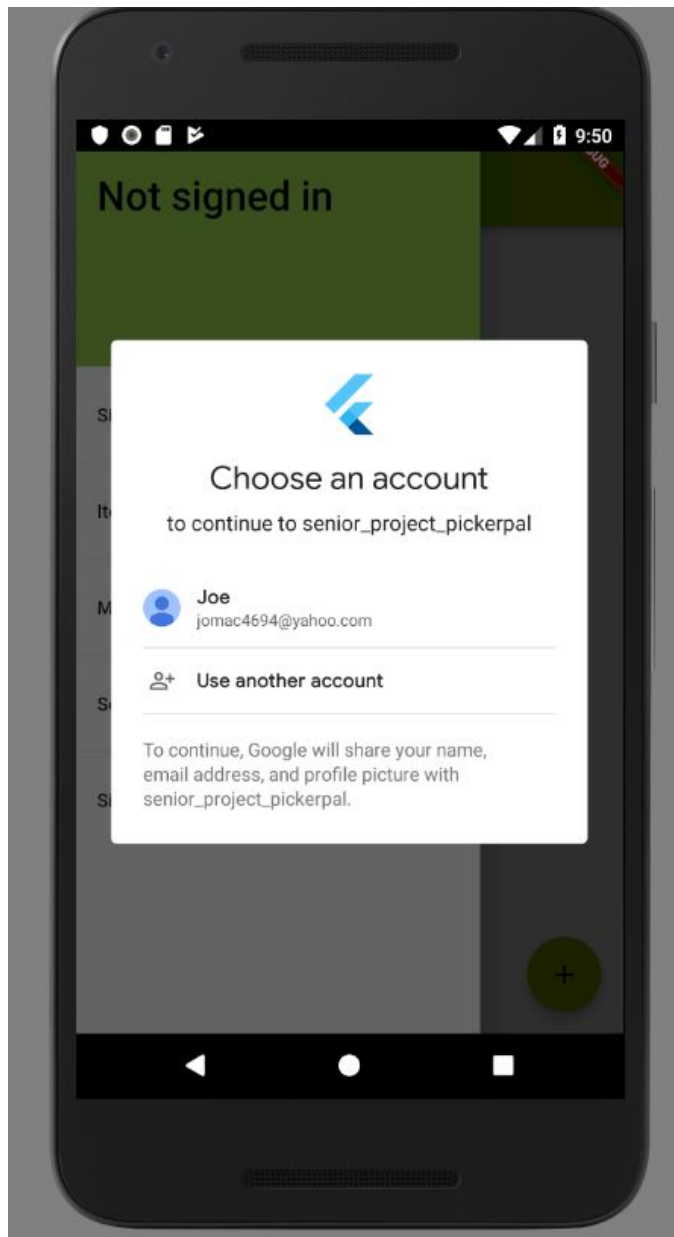
## Design Mockups - Entry Description

The entry description will be accessible by clicking on one of the listing tiles. It will provide more detailed information about the listing including the user who posted it and its location.



## Design Mockups - Login

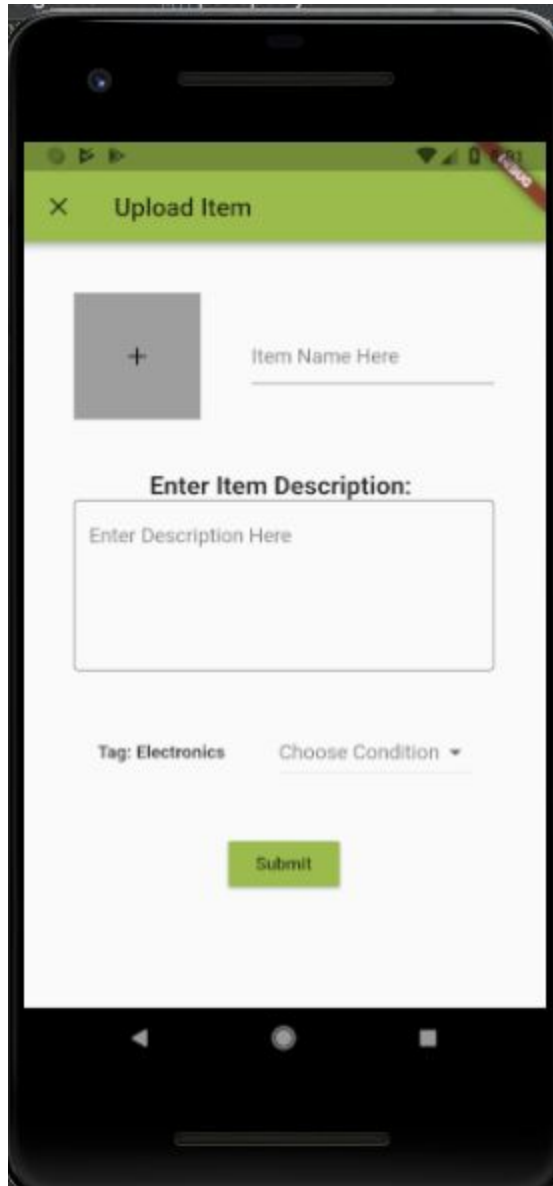
Users will be able to sign in with their Google account by accessing the right drawer and clicking "Sign in with google". They will then be prompted with a screen similar to what is shown below.





## Design Mockups - Upload Item

Users will be able to upload an item, if they are signed in, and depending on what category they pick a tag will be dynamically generated. This will require access to the camera so that they can upload or take a picture of the item they wish to put up.



The image shows a mobile application interface for uploading an item. The screen has a green header bar with a close button (X) and the title "Upload Item". Below the header, there is a grey square with a white plus sign for uploading a photo, followed by a text input field labeled "Item Name Here". Underneath is a section titled "Enter Item Description:" with a large text area labeled "Enter Description Here". At the bottom, there are two fields: "Tag: Electronics" and "Choose Condition" with a dropdown arrow. A green "Submit" button is located at the very bottom of the form area.

## Screen Navigation

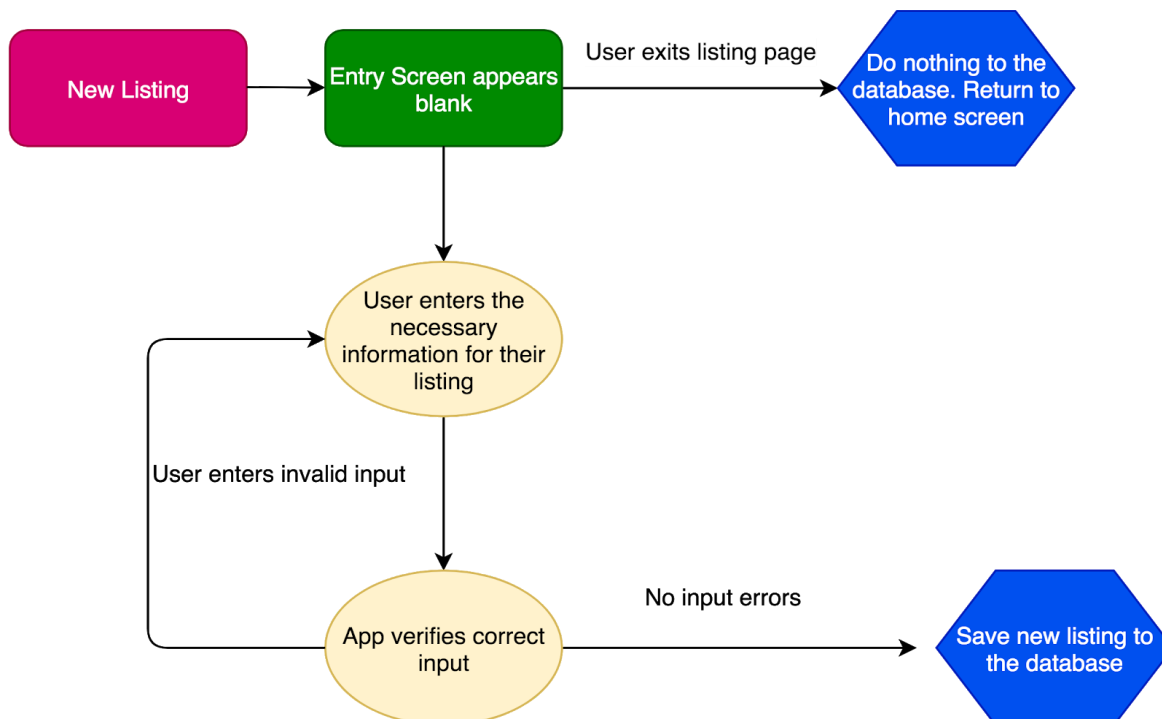
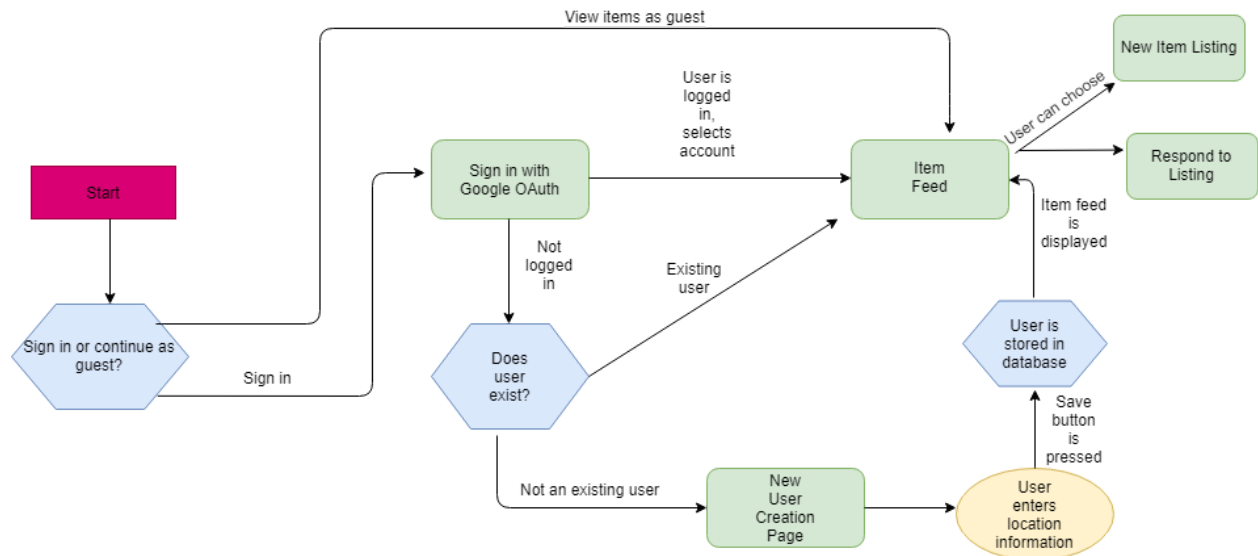
When a user initially opens the app, they will be presented with a dynamic feed of listings ordered by closest proximity to the users location. From this screen, users will be able to do a number of tasks. Using the drawer on the right, users will be able to login in through Google, access account settings, view their own listings, and sign out of their Google account. A search icon will also appear in the top right corner to access search functionality. Also, there will be a floating action button that can be used to submit a new listing.

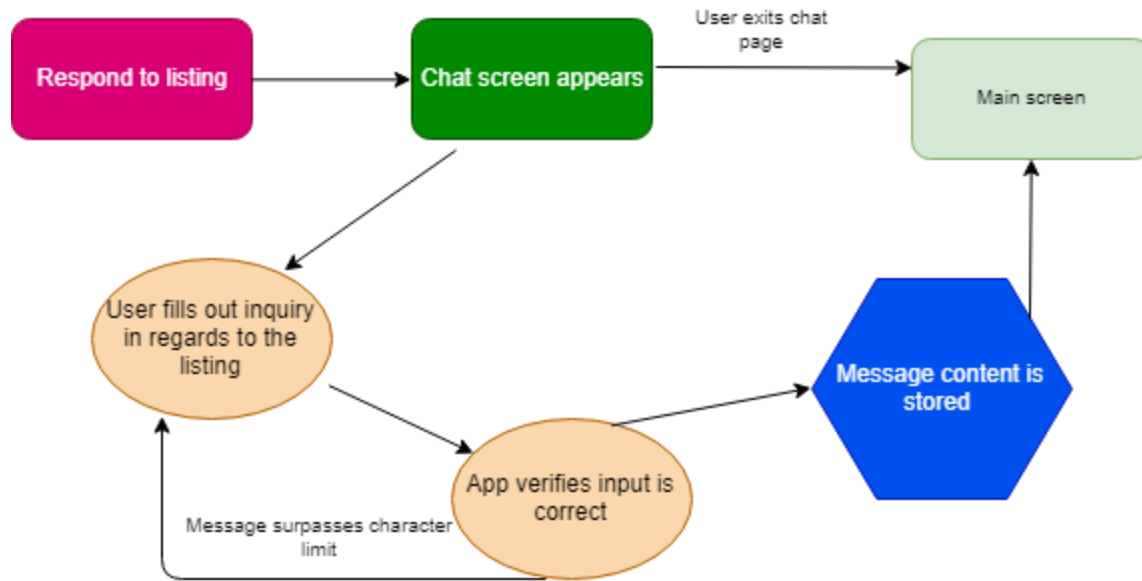
The dynamic feed will include listings submitted by other users. Each listing will contain an icon indicating the type of items, a small description, and a chat icon. If necessary, a user can click on the chat button to get any information about the items or setup a time to get them. When you click on a listing, it will present you with another screen that shows a picture of the items, where it's located, and a more detailed description.

The listing entry page will appear when the floating action button is pressed. From here, users can enter information about the trash they are trying to get rid of. This includes a title, description, location, and a category. When a users submits an entry, they will be directed back the home page.

The user settings page will be accessible through the drawer on the home page. Users will be able to view/edit any information about their account on this page. Settings such as app theme and others will be included. Also, users will be able to manage settings for receiving alerts about listings based on any tags of interests that they specify to the app. Users will also be able to view any of their own listings on a separate feed by pressing the "my items" options in the drawer.

# Flow Diagrams





## Technology Stack

Our technology stack makes use of Flutter/Dart for the front-end and Flask/Python for the back-end. The app's front-end is being designed primarily in Android Studio. However, parts of the front-end and back-end are being developed in VSCode. Testing is done using the Android Emulator.

We chose the Flutter framework for our front-end because of its simplicity and ability to quickly produce an interface that works with a RESTful API. Also, since the scale of the project is not large, Flutter was our best option. If the project was going to be used on a larger scale, React Native may have been more appropriate. Lastly, we chose Flutter because a few of our team members were already familiar with it.

We used Google Sign-in and OAuth for login because it's secure and easy to implement. The user's email is then used as a key into our database of profile entries.

Our backend is done using Python and the Flask microframework, which implements a RESTful API. We then use HTTP requests to produce our data on the front-end as needed. Similarly to our front-end choice, we chose Python because of simplicity and familiarity, as well as its vast library of third-party addons.

Our database is hosted using AWS. We chose this because it was provided to us by our instructor and it is a well known service. The server itself requires less than a GB of RAM to run, and is running Ubuntu 18.04.1 LTS.

## Backend Information

Our backend makes use of the Flask microframework and Python. The Python script communicates with the AWS hosted MySQL database using a series of HTTP GETs, POSTs, and DELETEs. The backend is responsible for storing and managing all information on users and listings.

The backend is also responsible for querying and ordering listings based off its distance from the user. This lets the front-end only have to worry about having to present the data and not need to do any extra sorting.

```
SELECT
  id, (
    3959 * acos (
      cos ( radians(78.3232) )
      * cos( radians( lat ) )
      * cos( radians( lng ) - radians(65.3234) )
      + sin ( radians(78.3232) )
      * sin( radians( lat ) )
    )
  ) AS distance
FROM markers
HAVING distance < 30
ORDER BY distance
LIMIT 0 , 20;
```

This will be achieved by storing lat and longitude as columns in the database to be used in the aforementioned distance computation.

## Authentication and Security

Our app's user authentication is done using OAuth. The user simply needs to login using their Google email and password, the rest is then handled by Google. We chose this method for its convenience, since people don't like being forced to make a new account.

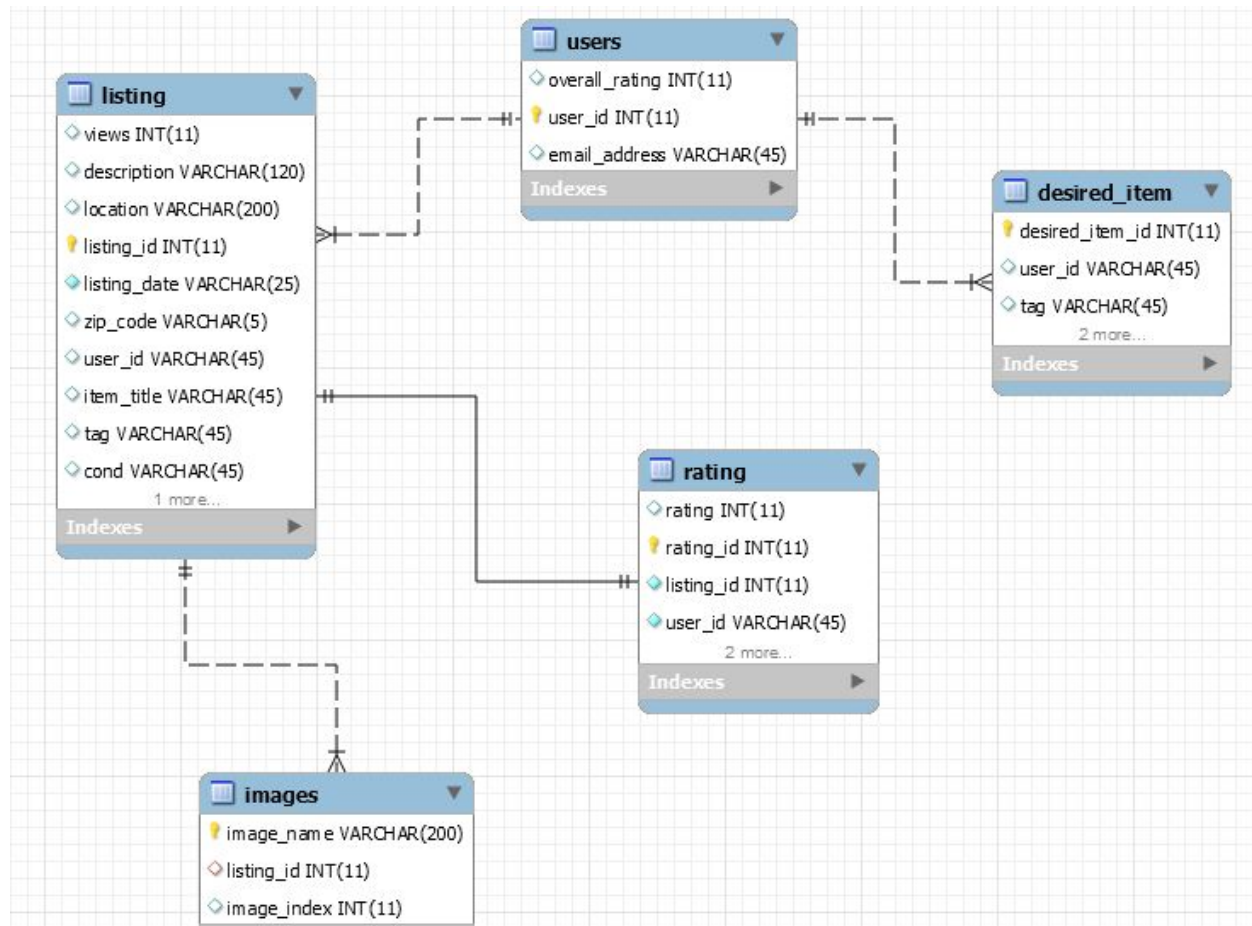
## Input and Output

Input to the app is primarily accomplished through virtual buttons on the touch screen, the device's physical buttons, and the device's virtual keyboard. The camera is also used for input to take a picture for a listing entry. Other input will be delivered through user interactions with push notifications and snack bars.

Output is provided as a dynamic feed of entries sorted by what is closest to the user's location. The app will also create push notifications when a chat message is received, or when a listing of a user-chosen category is posted in the user's area. Snack bars will notify the user when a listing could not be posted or a chat message could not be sent.



## Database Schema



The database itself consists of five main tables: listings, users, ratings, desired\_item, and images. Due to the nature of the relationships between each table, no bridging of tables is necessary. One user has many listings and desired\_items, while each listing one rating and many images. This will allow the tables to be joined in circumstances where returning information from multiple tables is necessary.

As more features are implemented, the structure of the database itself will grow, and this ER diagram is subject to change.

## Restful Endpoints

### **GET /users/{user\_id}**

Returns a JSON containing information corresponding to the user\_id if it exists.

### **GET /ratings/{rating\_id}**

Return a JSON containing information corresponding to the rating\_id if it exists.

### **GET /ratings/{user\_id}**

Return a JSON containing all ratings for a given user\_id if it exists.

### **GET /listings/{listing\_id}**

Return a JSON containing information corresponding to the listing\_id if it exists.

### **GET /listings/{zip\_code}**

Return a JSON list containing JSONs for listings within a certain zip\_code, if the zip\_code exists.

### **GET /listings/{tag}**

Return a JSON list containing JSONs for listings within a certain tag, if the tag exists.

### **GET /listings/{condition}**

Return a JSON list containing JSONs for listings within a certain condition, if the condition exists.

### **GET /listings/{radius}**

Return a JSON list containing JSONs for listings within a certain radius, if the radius is valid.

### **GET /desireditem/{user\_id}**

Return JSON containing all desired items associated with the user\_id if any exist.

### **POST /users/new**

Creates a new user profile using the information within the JSON. Returns the newly created user.

### **POST /listings/new**

Creates a new user listing using the information within the JSON. Returns the newly created listing.

### **POST /ratings/new**

Creates a new user rating using the information within the JSON. Returns the newly created listing.

### **POST /desireditem/new**

Creates a new desired item using the information within the JSON. Returns the newly created desired item.

### **DELETE /listings/delete/{listing\_id}**

Deletes the database entry associated with listing\_id, if the listing\_id exists. Returns the deleted listing.

### **DELETE /users/delete/{user\_id}**

Deletes the database entry associated with user\_id, if the user\_id exists. Returns the deleted user.

### **DELETE /ratings/delete/{user\_id}**

Deletes the database entry associated with rating\_id, if the user\_id exists. Returns the deleted user.

### **DELETE /desireditem/delete/{desired\_item\_id}**

Deletes the database entry associated with desired\_item\_id, if the desired\_item\_id exists. Returns the deleted user.

## Team Responsibilities

Joseph Whittier: Back-end design/development

Ryan Beebe: Front-end design/development

Joseph Mcilvaine: Front-end design/development

Matt Degenaro: Back-end design/development

Joe Antaki: Back-end and front-end design/development

Gianluca Solari: Front-end design/development and logo development

Sean Spencer: Back-end design/development

Manoj George: Front-end design/development/morale booster

## Mid-Assessment Goals

Our goals for the mid-assessment are as follows:

- Implement the ability for users to receive alerts for a listing based off of some tag/keyword(s).
- Fully display all information associated with some listing on the front-end of the application.
- Implement the remaining endpoints on the back-end.
- Implement remaining functions on the front-end for communicating with the back-end.
- Implement a more robust search function.
- Choose a design for the applications splash screen.
- Implement a private messaging system so pickers and people trying to get rid of trash can communicate.