CATNAPP

Android Programming Final Project – Fall 2020

Abstract

CatNapp allows a user to sign-in, find cats based on their preferred "cat-egory", save these cats to their account, and share them with their friends via text, email, and social media.

Jake Grogan, Payton Suiter

jakegrogan@utexas.edu, suiterpayton@utexas.edu

Detailed Overview

This section describes each page of the application in detail.

Landing page

Upon loading the application, the user is directed to the main landing page (after sign-in if required). Here the application will display the username of the currently signed in user, as well as a button to access favorites and a pop-up menu to sign in or out. *Figure 1* shows the basic landing page as seen on start-up. *Figure 2* shows the pop-up menu with the "Sign Out" option as there is currently a user signed in. *Figure 3* shows the category selection spinner and available options.

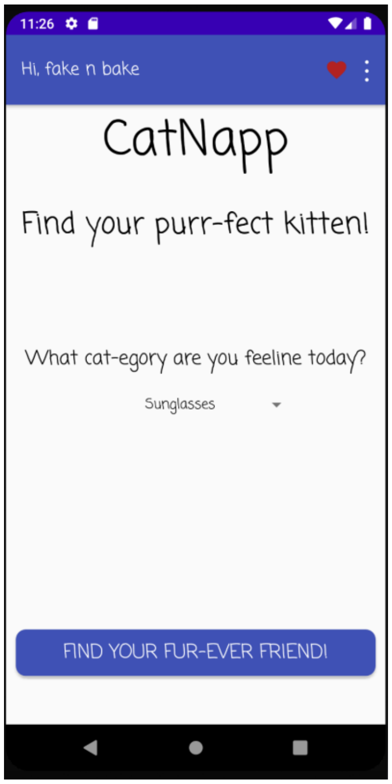

11:26
Hi, fake n bake
CatNapp
Find your purr-fect kitten!
What cat-egory are you feeline today?
Sunglasses
FIND YOUR FUR-EVER FRIEND!

Figure 1

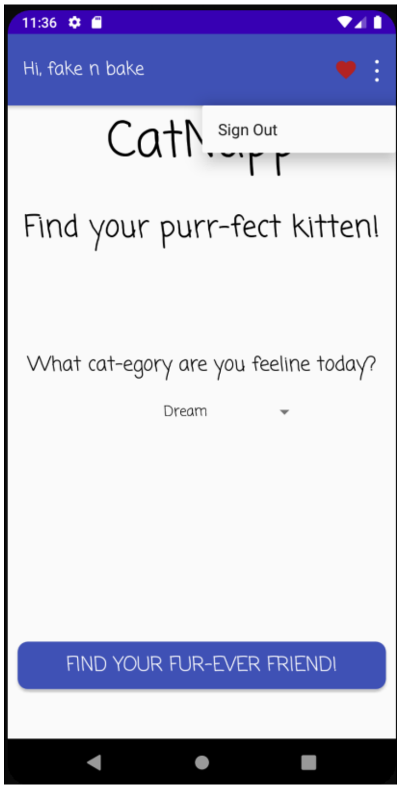

11:36
Hi, fake n bake
Sign Out
Find your purr-fect kitten!
What cat-egory are you feeline today?
Dream
FIND YOUR FUR-EVER FRIEND!

Figure 2

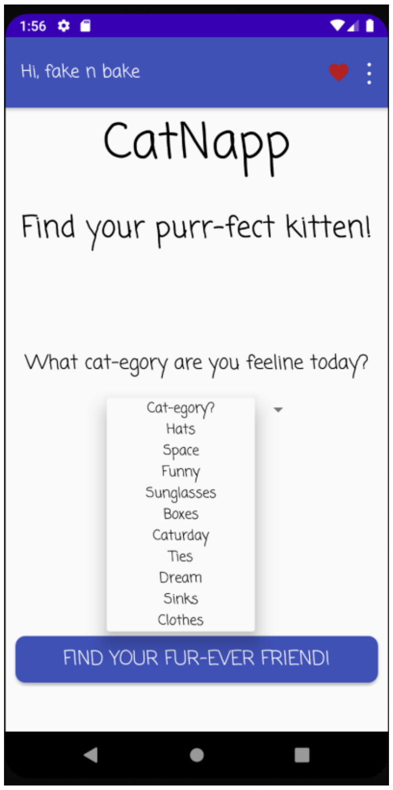


Figure 3

Cat selection page

This page contains the meat of the application, it is where users can browse cat images returned by the API, view details, and select favorites. This page also contains the action bar showing the signed in user as well as buttons for favorites and the pop-up menu. *Figure 4* shows the cat browsing view, this view will load 10 cats from the API and display them to the user. The user can swipe down to refresh the view with new cats. *Figure 5* shows the cat details view, which shows an enlarged image and any comments the user has entered about that cat. Of note, in order to edit comments, the user must access the details view from the favorites page.

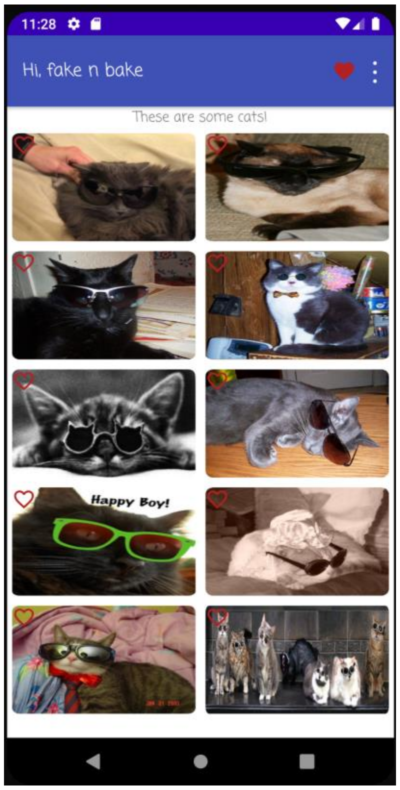


Figure 4

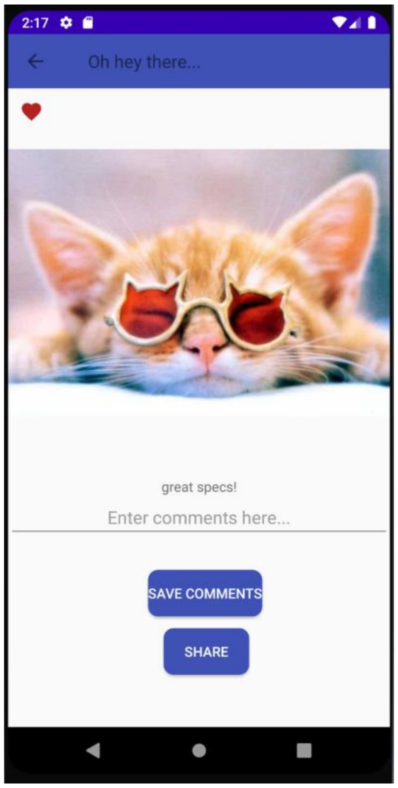


Figure 5

Favorites page

The favorites page allows users to view cats they have previously saved. The images are loaded from Firestore, so the favorites will persist between sessions. *Figure 6* shows this view. Additionally, a user may use the pop-up menu to sign out (removing all images from the view) and sign in with another account to load a different set of images. *Figure 7* shows the details page loaded from the favorites view. This looks similar to the view from select cats except that now the user may enter comments on the cat. These comments are saved in Firestore and will be displayed when the details page is loaded in future sessions.

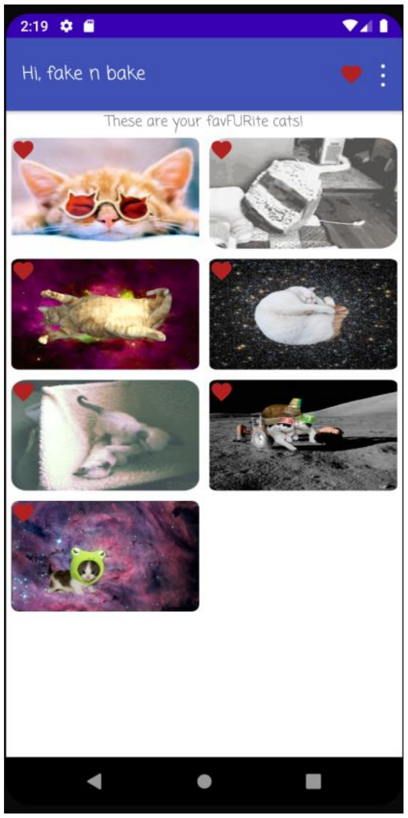


Figure 6

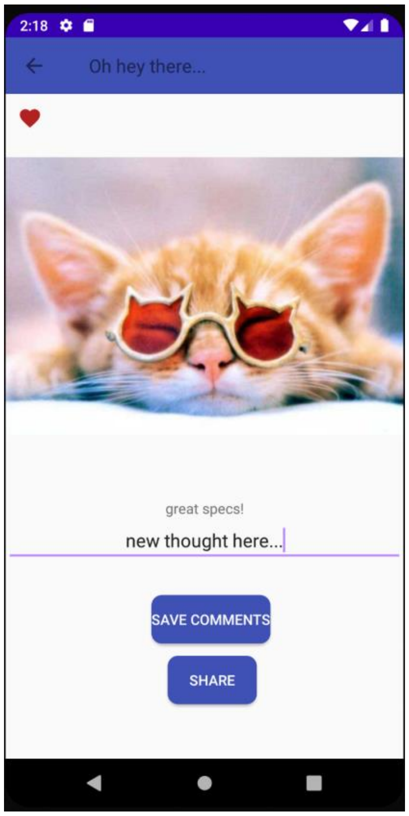


Figure 7

Share feature

As discussed above, the details page provides additional information on a cat image. It also allows a user to share an image using the share button, which launches the intent shown in *Figure 8*. Here the user can select the desired platform for sharing. As show in *Figure 9* and *Figure 10*, the user is able to send both the image and a link to the image (Twitter image not shown as the emulator load time was incredibly long).

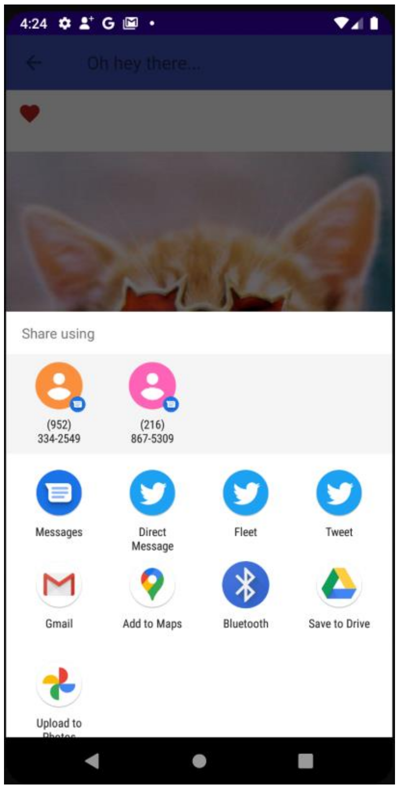
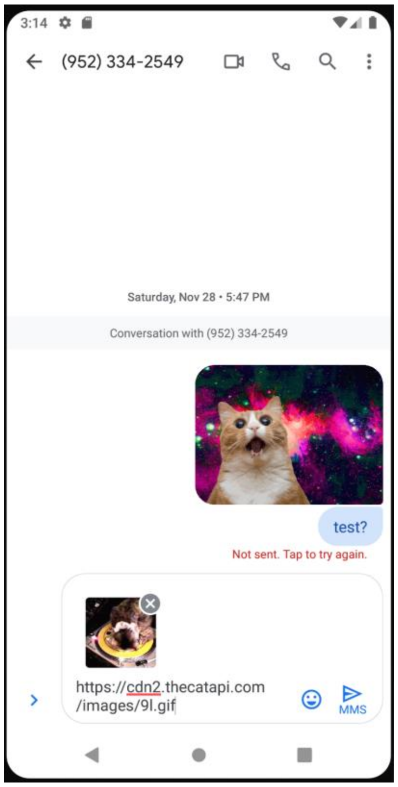


Figure 8

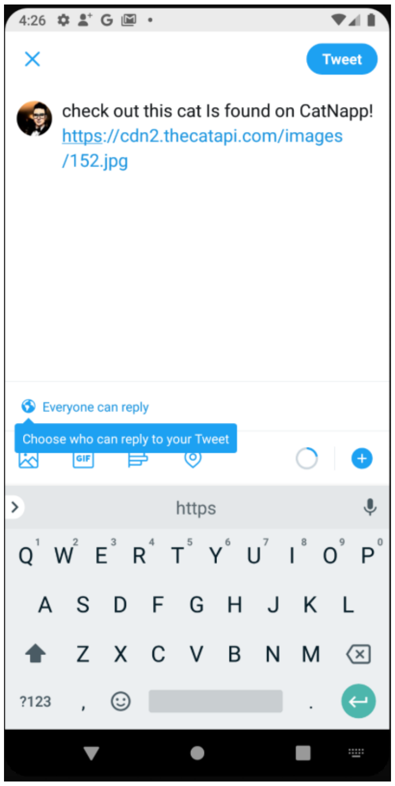


test?

Not sent. Tap to try again.

https://cdn2.thecatapi.com/images/9l.gif

Figure 9



check out this cat Is found on CatNapp! https://cdn2.thecatapi.com/images/152.jpg

Figure 10

APIs and Android Features

API

The primary functionality of our application was enabled via TheCatAPI (https://thecatapi.com/), which provided thousands of images and gifs organized by category, breed, etc. Specifically, our application allows a user to search for cat images by a user specified category. Our implementation allows for both images and gifs to be displayed.

Action bar

We also utilized an action bar to enable various functions. This is present at each window of the app and allows the user to manage the signed in user as well as navigate to favorites. Specifically, the bar contains a title to indicate who is currently signed in or if there is no user signed in. It also contains a favorites button to allow the user to navigate to their saved favorites if they are signed in. Finally, the action bar contains a menu button that allows users to sign in and out.

Menu pane for sign in and out

As mentioned above, a notable feature of this application is a menu popup that allows users to manage which account is signed in. If a user is signed in, this menu will provide an option to sign out the current user. Once signed out, clicking on this menu will provide a means for a new user to sign in. This functionality allows for users to sign in and out of the application while it is in use and prevents the need to restart the application to sign in a new user. Of note, if a user signs out while on the favorites page, their favorites will disappear as no user is currently signed in (images will reappear upon sign-in).

Libraries

Retrofit

We used Retrofit to build our API calls and parse the returns. This was similar to how we used Retrofit throughout this course, but we ran into a particular issue with GSON on how to parse returns that were a combination of Objects and Lists. We were able to work around this by creating separate data classes for the Lists and passing them into the return structure as an object. We also utilized the @Expose feature of GSON to choose which values to parse and which to ignore, which helped in removing values that we didn't need and that were breaking the parse.

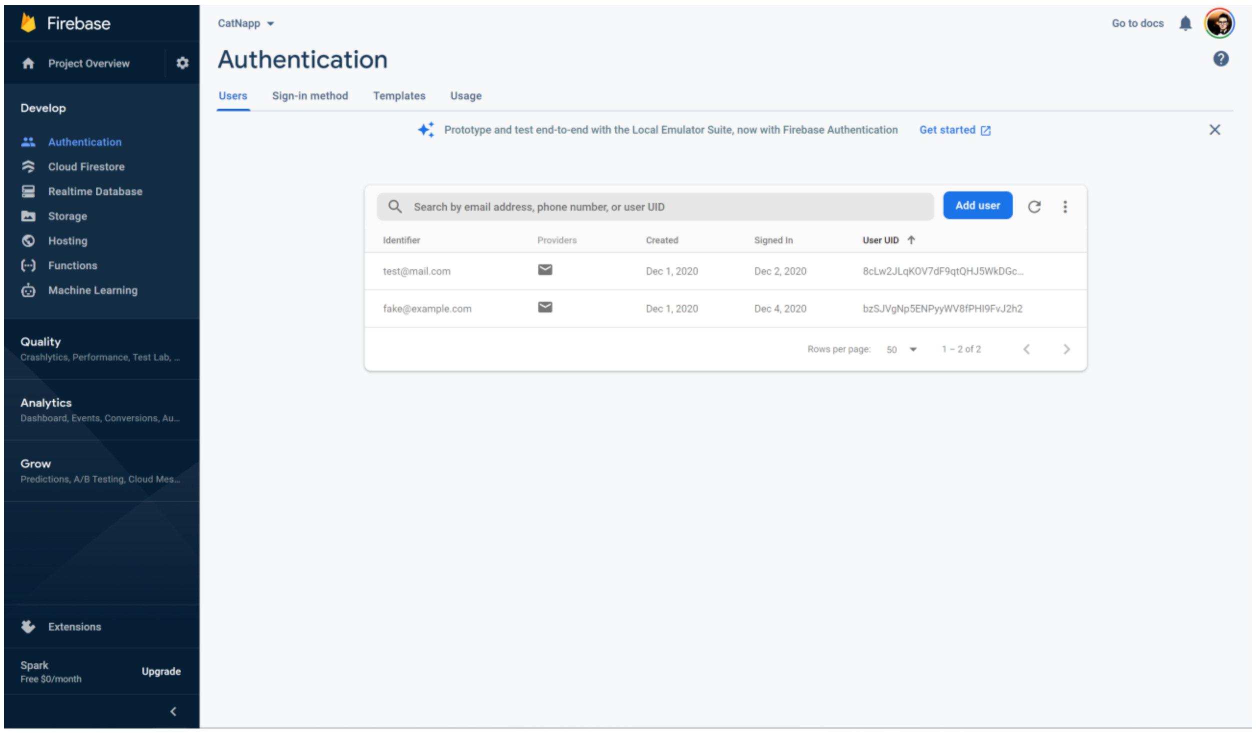
Glide

We used the Glide library to load images from a URL into our Recycler view. We also used it to do cleanup of the images, so they all displayed uniformly. We also used the transform feature to round the corners of the images to give them a modern look.

Firebase Auth

A major library we used was Firebase Authentication. This allowed users to sign-in and save images between sessions on the app. If there was no user, one could be created, otherwise the user would login as normal.

A challenge was allowing users to dynamically sign in and out of the application. We wanted users to be able to change accounts or sign out from within the application instead of needing to restart the app to be prompted for the sign in option. For simplicity, we only enabled email and password sign in, which allowed us to save user data and showcase that functionality for the purpose of this report. *Figure 10* shows our Firebase Auth console.



Authentication

Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication

Search by email address, phone number, or user UID

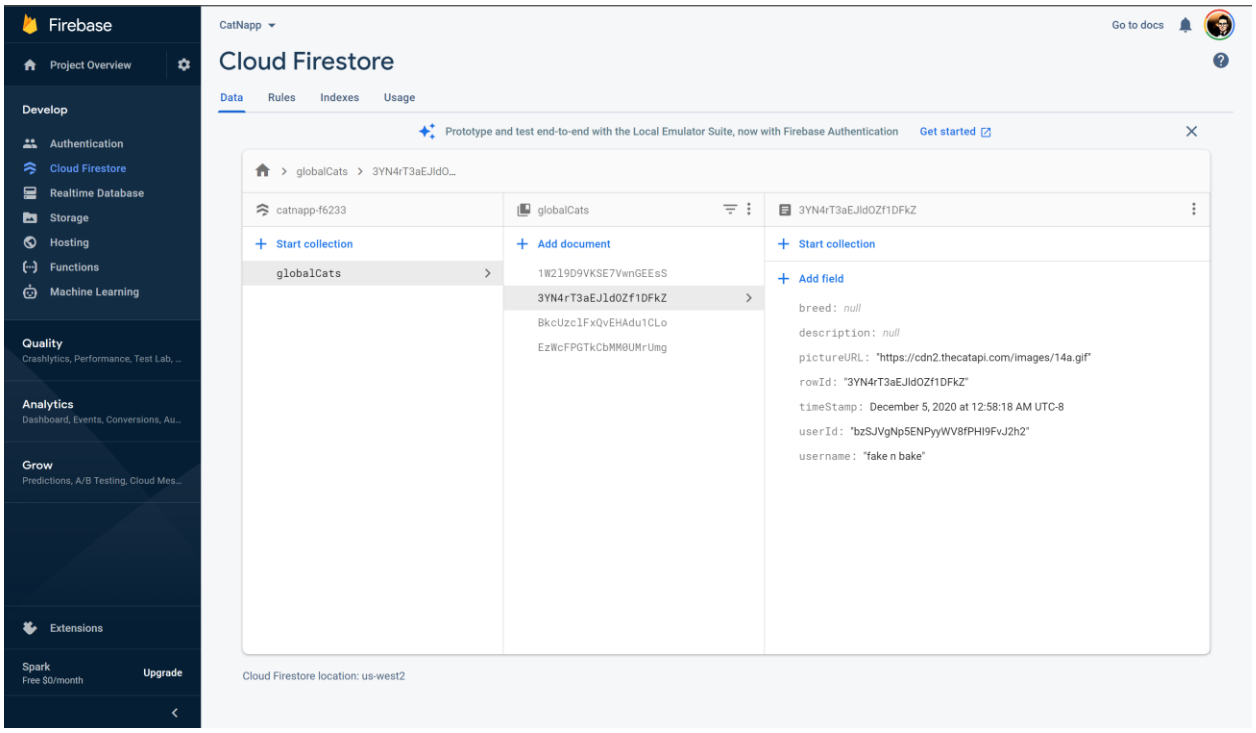
Identifier	Providers	Created	Signed In	User UID ↑
test@mail.com	✉	Dec 1, 2020	Dec 2, 2020	8cLw2JLqKOV7dF9qtQHJ5WkDGc...
fake@example.com	✉	Dec 1, 2020	Dec 4, 2020	bzSJVgNp5ENPyyWV8fPHI9FvJ2h2

Figure 10

Firestore

Another third-party service we used was Firestore. This was used in conjunction with authentication to allow us to save user favorites to a persistent database. This allowed us to keep track of user favorites between sessions, which provides an improved user experience. *Figure 11* shows the database implementation.

One of the challenges we faced with this was how we were wanting to save the image within the database. We wanted to save the actual image itself, but we decided that saving the image URL provided a better experience for two reasons. Firstly, we found that saving an image resulted in a much greater delay in storage response time, such that if a user added a favorite then immediately deleted it, they could end up making a deletion request to an image that was not yet saved, crashing the application. By saving the URL and fetching the image on bind, we were able to reduce latency and avoid this issue. Secondly, as CatNapp does not retain ownership of these images, it seemed prudent to hold a link to the image so that the rights of publishing remained with the original owner.



```
globalCats

1W219D9VKSE7VwnGEEsS
3YN4rT3aEJldOZf1DFkZ
BkcUzclFxQvEHAdu1CLo
EzWcFPGTkCbMM0UMrUmg

breed: null
description: null
pictureURL: "https://cdn2.thecatapi.com/images/14a.gif"
rowId: "3YN4rT3aEJldOZf1DFkZ"
timeStamp: December 5, 2020 at 12:58:18 AM UTC-8
userId: "bzSJVgNp5ENPyyWV8fPHI9FvJ2h2"
username: "fake n bake"
```

Figure 11

Noteworthy Features

GridLayout for RecyclerView

One of the noteworthy parts of our UI was our use of the GridLayout for our RecyclerView. We wanted to display a grid of images, as we felt it was a more desirable user experience. In conjunction with this, we limited returns to just 10 images, ensuring a clean presentation with no scrolling required. Since the goal was to clearly present the best images, we felt this was the most effective way. For a user to see new images, they simply need to swipe down to refresh the view.

Persistent Favorites

For the backend, Firestore was also one of our most noteworthy backend features. This allowed us to save favorites for a user to a persistent database and subsequently fetch those images for a user in later sessions. Firestore allowed us to provide a significantly improved user experience, as now images could be saved and shared at a later time, removing the need for a user to re-discover their favorites.

Image Sharing

Another unique feature of this application is the ability to share cats with others. Namely, we implemented a sharing intent that allows user to share the image and their comments via various platforms, including email, text, and social media. We personally feel this is the most exciting feature, as it takes the application from simple amusement to a platform for sharing the fun with others. This is more engaging to the user and provides a semblance of "purpose" to the application, as users are able to share the results of their exploration.

Custom UI features

We also enabled unique UI features to provide a modern experience while using our application. Specifically, images and buttons are enabled with rounded corners, the font is changed to match the playful nature of the application, and the color scheme is changed to differentiate the application from a base android app. We also used as many cat puns as we could, because punny is the best kind of funny!

Future Features

This section documents follow on features that we did not have time to implement but would handle if we continued to develop the app. These actions are currently handled as edge cases with prompts directing users on how to complete their desired action in a supported manner. As such, these features represent an improvement to available functionality and are not considered bugs.

1. Allow users to add/remove favorites from cat details view
2. Allow users to add comments from the details view when a favorite is opened from the CatAdapter. Currently only able to update comments if opened from Favorites fragment.
3. Allow users to update the "cat-egory" from the cat selection page

Lessons Learned

One of the most interesting features we implemented within our app was the ability to share a cat image via text/email or to a social media account. This was very new to us but sounded like an interesting feature to attempt. The integration required reading a lot of Android documentation, and it took some time for us to figure out how to test it correctly within the emulator, but we were able to make it work and feel it provides a much more engaging user experience.

One of our largest challenges occurred with our API. At the start, it took some time to figure out how to integrate an authorization key within the API call, so it would correctly retrieve the results. Once we started getting 200’s, we thought everything was fine; unfortunately, we ran into an issue with GSON parsing the return information. We were using our previous homework as a guide for how to correctly use Retrofit on our result data but were having issues with non-uniform return types (Objects and Arrays in the same .json). It took hours to figure out what was wrong, because every change we made within the API also affected the Cat data class or the repository call. We were eventually able to match the Retrofit expectation to the return using additional data classes and the @Expose feature.

Total lines of code

Kotlin: 889
XML: 413

The way we went about counting these lines is first running the command to count all of them, then going through the Kotlin and/or XML section and subtracting lines that were either boilerplate or not authored by us.