Final Project Report

I. Application Title

ShareIn

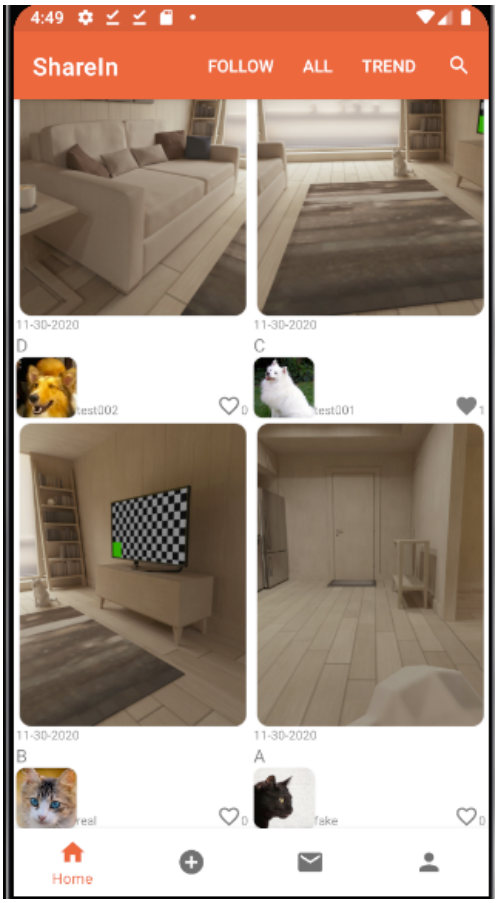
II. Group Member Information

Name	EID	Email
Yue Pan	yp3723	panyue@utexas.edu

III. Application Description

ShareIn is a social networking application which allows users to post and share their stories and photos with other users. User can also become friends with other users by following them and sending them messages.

IV. Application UI



Home Page of ShareIn

V. Key Functionality

1. User is required to login with his/her email and password to be able to use the application. If this is the first-time login, an alert box will pop up to ask user if he/she would like to reset the username. The default username is the part which is in front of the "@" of user's email. For example, if user's email is fake@example.com, the default username is fake.
2. User can set up profile photo by choosing a photo from library, cropping it and uploading it. After successfully uploading the profile photo, a toast message will pop up indicating upload is successful.
3. User can create new post. User is able to:
    - Enter a title
    - Enter post content
    - Take photos and upload
    - Choose a background song from three songs available
    - Enable/disable location
4. User can short click a post to view the post. When viewing a post, user can:
    - Like/unlike the post
    - Comment to the post or delete the comment
    - Clicking on the profile photo will direct the user to the home page of the post owner
    - Follow/unfollow the post owner
5. User can longer click the post to do the following if the user is the post owner:
    - Edit the post
    - Delete the post
6. Go to user's me page ("Me" in the bottom navigation bar), by default we will see all user's posts.
    - Clicking on "POSTS" will refetch all user's posts
    - Clicking on "LIKED" will refetch all user's liked posts
    - Clicking on "Following" or "Follower" will direct user to a list of following/followers
7. Go to user's home page ("Home" in the bottom navigation bar), by default we will see posts from all users ordered from most recent to least recent. Clicking on "ALL" in the top menu bar will perform the same action. Clicking on "FOLLOW" in the top menu bar will refetch all posts from whom user is following. Clicking on "TREND" in the top menu bar will refetch all the posts and order them from most liked to least liked. After clicking on the search icon in the top menu bar, a search box will pop up and user can search for the posts that contain the typed-in keyword in either post title or post content.
8. User can view other user's me page and do the following:
    - Follow/unfollow the other user
    - Send messages to the other user
    - View other user's posts
9. Clicking on "Message" icon in bottom navigation bar will direct user to the message inbox. It will display a list of messages received by the user. Only one message per sender will be displayed here and this message displayed is the latest message received by user. Clicking on the row will direct the user to a message page with a specific sender. User can send message to that sender there.

VI. APIs

1. Google Firebase Authentication
2. Google Cloud Firestore
3. Google Firebase Storage

VII. Android features

1. I use alert box extensively whenever I need to prompt for user input. I find it super convenient to use. Without the use of alert box, I could have ended up writing a lot more fragments with each of them does only minimal amount of work. Moreover, with the use of alert box, I am able to make minimal impact to the activity lifecycle. It's a lot easier to handle than using fragments.
2. I use navigation map extensively as well to handle movement among fragments. It's also very convenient to use and easy to define navigation action and arguments to be passed among fragments.
3. I use Android's location service for the implementation of showing location of the post owner.

VIII. Third party libraries and services

1. Glide is used to fetch images. It is quite convenient to use because it allows direct fetching and binding to image view.
2. Android image cropper is used for cropping profile photo uploaded by user. It has very detailed instruction regarding how to set it up in build files as well as how to properly incorporate it into the application. I didn't run into any issues when setting it up so I think it is very user friendly.
3. ViewPagerIndicator is used to implement the circle indicators below the image slider in the post view. There are a lot of detailed set-up and incorporation instructions online so it's also very convenient to use by simply following the instructions.

IX. Noteworthy UI/UX

1. I use RadioGroup and RadioButton when implementing the background music selection functionality. Radio button is the best fit because it allows only one selection and the item selected is clearly indicated. Its onCheckedChangeListener is also very helpful for app to play/pause the music upon change of music selected.
2. I use ViewPager2 together with RecyclerView adapter to hold the image slider in post view. An image slider can hold multiple images in one single viewpager2 view and user can view different images by simply swiping left and right. This allows a more graceful and space-saving way to display multiple images.
3. As mentioned earlier, I found alert box very convenient to use. I use it extensively when I need some user input or some user confirmation before performing certain action. Alert box allows me to write fewer fragments and to make minimal impact to the activity lifecycle.

X. Noteworthy backend/processing logic

1. Create different data classes to avoid data race:

There are a few places in the app where data race could possibly happen. For example, one of the functionality is allowing user to like/unlike a post. If we simply set a field "likeCount" to count how many likes are associated with a post in the data class Post, data race could happen when multiple users click "like" to one post at the same time. As a result, some "likes" will be missing and likeCount is likely to be inaccurate. In order to avoid this kind of situation, I created a data class Like to record the user uid as well as which post he/she likes. When a user likes a post, one entry with these fields will be created. And when he/she un-likes this post, this entry will be removed. When counting total likes of a particular post, we just need to fetch the database by post id and get the result size. Similar logic also applies to follow, comment, and message. The general idea is whenever multiple users are able to access some shared resources simultaneously, we need to watch out for data race.

2. Going back and forth from following/follower list to Me page:

This was one of the bugs I encountered when I was doing the project. One of the functionalities I intend to implement is to allow the logged-in user to view the Me page of other users (either following or follower). In other words, if the logged-in user(A) clicks on either following/follower on his/her Me page, a list of following/follower would be displayed. If the user clicks on one of the rows, he/she would be directed to the Me page of the user(B) that he/she just selected. The issue I encountered was when I intended to go back from other B's Me page to the following/follower list of the A or when I would like to go further down to other B's following/follower list to view the following/follower of B, the list wouldn't pop up the correct content. The bug was caused by the fact that I only have one mutable live data list to store all the following users and one mutable live data list to store all the follower users. When I landed on B's Me page, the following and follower lists were both updated to store B's stuff. And when I went back from B's Me page to A's following/follower list, A's stuff is already gone. And B's following/follower list would be displayed. The best solution I could think of is to maintain a stack of user. For example, when leaving A's following/follower list to get to B's Me page, A should be pushed to the stack. And then when going back from B's Me page to A's following/follower list, I just need to pop the user from the stack and fetch the following/follower list accordingly. This logic also works perfectly if I would like to go further down to the following/follower list of B.

3. Logic implementation for Trend:

One functionality implemented is allowing user to view all posts ordered from one with most like counts to one with least like counts. Since likes are maintained in a separate data collection, the logic here to sort all posts from most liked to least liked is worth a bit of explanation.
Since we are sorting all posts, we first need to fetch all the posts and then put all the post IDs in a list. Then we can use this list to fetch all the likes. Here we need to use whereIn to fetch since we are giving the database a list of post IDs to fetch. According to the Google firestore database document, we can only give a list of no more than 10 elements if we would like to use the whereIn clause. Therefore we need to chunk the list into pieces of size 10. Then we would fetch in a loop. After we are done with fetching, we can then start to count likes for each post and then sort the postsList from one with most likes to one with least likes.

XI. Interesting Things/Challenges

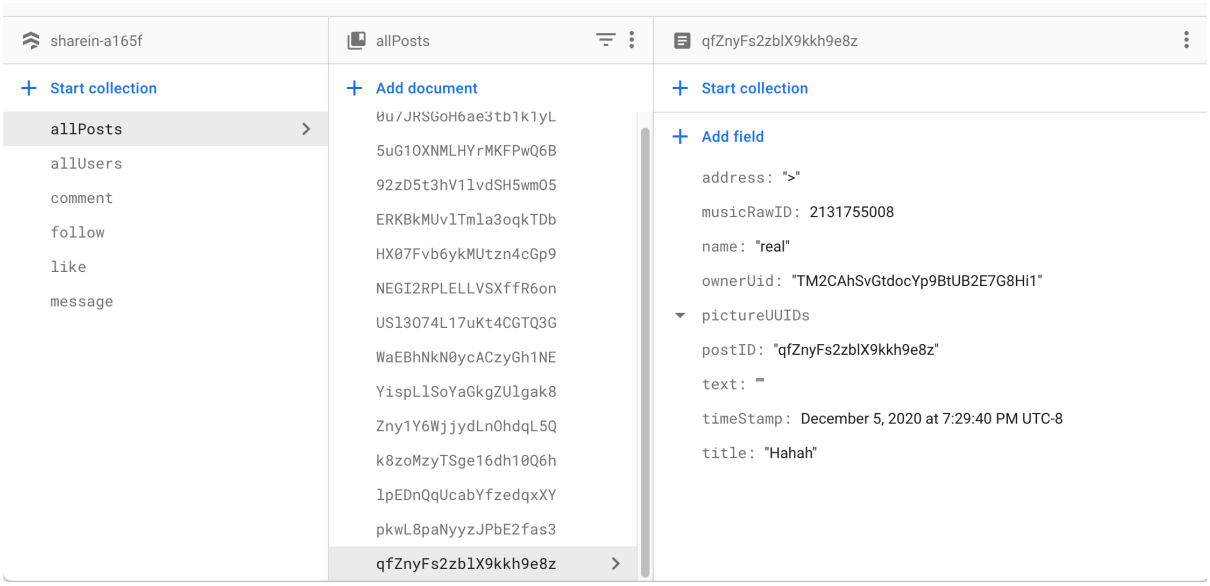
One interesting thing I learned from doing the project is activity's lifecycle. I remember that we talked about lifecycle at the very beginning of the class but somehow I didn't pay too much attention to it until I ran into troubles when implementing the back button logic in Reddit homework as well as when doing this project. What I learned is I should never make assumption of what stage of the lifecycle the activity/fragment is currently at and using Log.d to get the current stage is always super helpful. When I was doing the project, I ran into bugs once because I assumed the fragment was destroyed but it wasn't there yet (only the view was destroyed). Using Log.d helped me notice that I was making the wrong assumption and solved my problem.

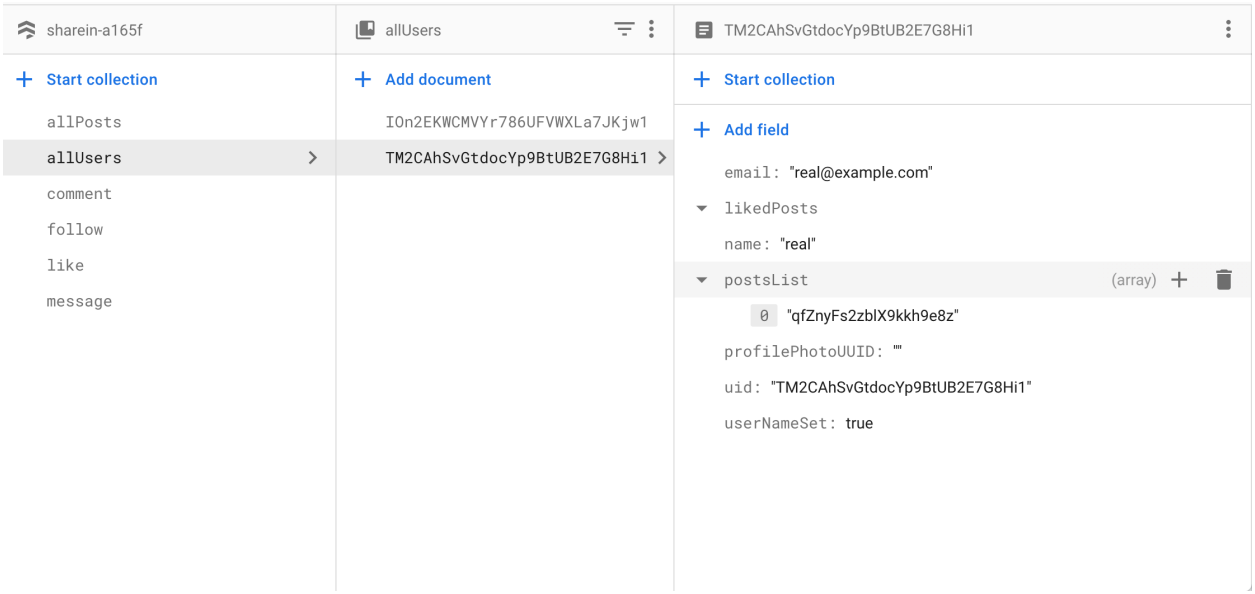
When doing the project, most bugs that I ran into was due to incorrectly dealing with database fetching result. For example, I have a user data class and user collection in database to store users of the application. Sometimes I need to fetch a particular user from the database to get his/her data so that I'm able to bind the data to view (i.e. Me page). I remember that there were a few times that I ran into user was null error. After doing a few log print out I noticed that I ran into these errors because I was trying to bind data to view before the fetch is completely done. The correct way to do this was to either handle data binding in the onSuccessListener or to handle it with the help of live data observer. Also I learned that I need to be very careful with observer. One mistake I made when doing the project was that I modified the data that I was observing in the callback which led to an infinite observing. I should never modify what I'm observing in the callback.

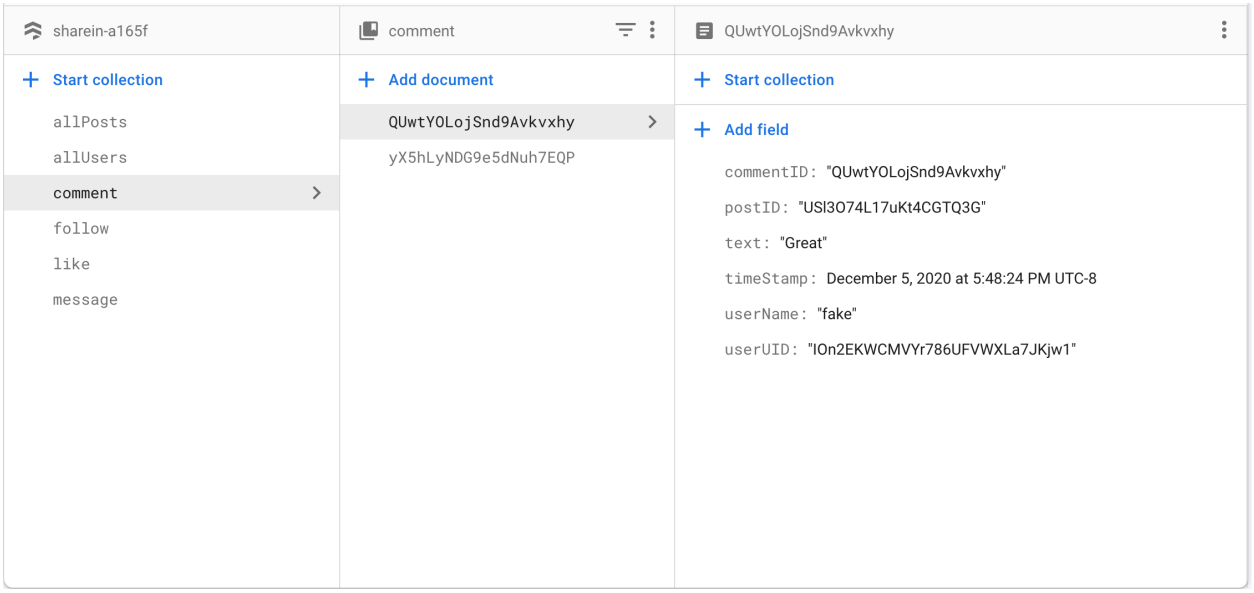
XII. Build instructions

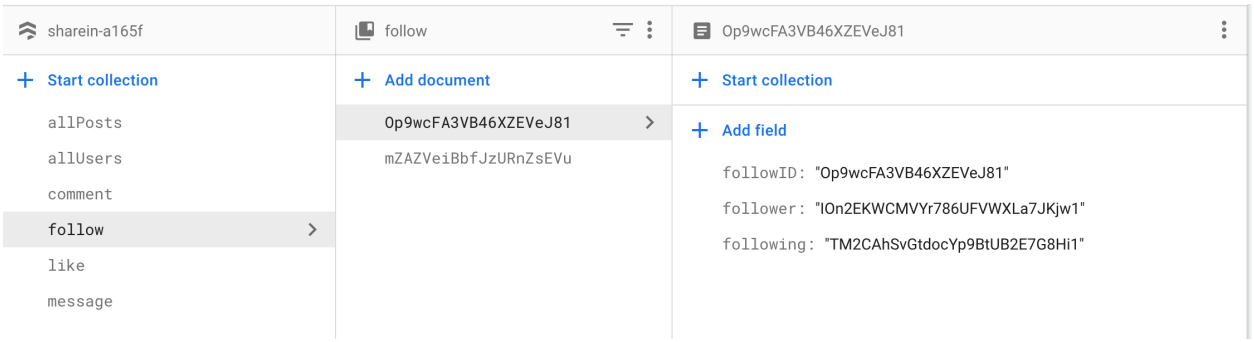
1. (Optional): To have a few photos in phone/emulator's local library in case you would like to test out the profile photo upload functionality.
2. (Optional): Turn on the location service of the phone/emulator. It is optional because even if the location service of the phone/emulator is not enabled when running the application, the application would ask you if you would like to enable location service and direct you to the setting page when you are trying to test out the location functionality in the application.

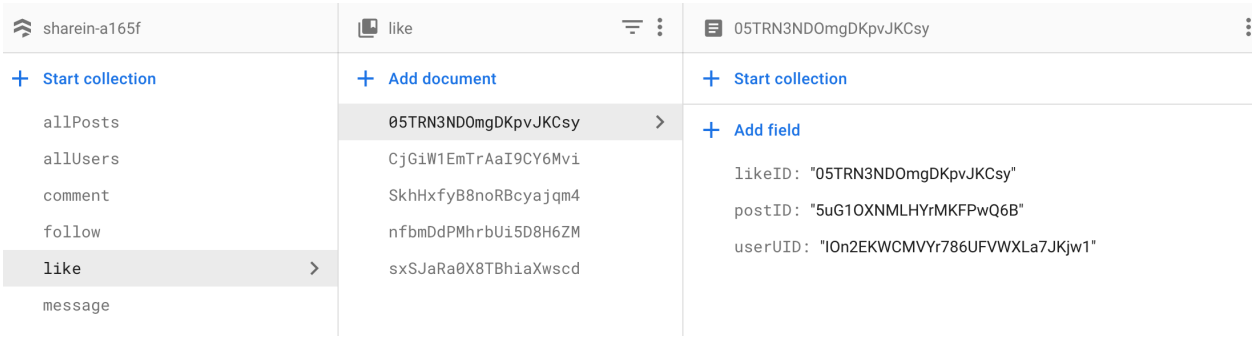
XIII. Database Schema

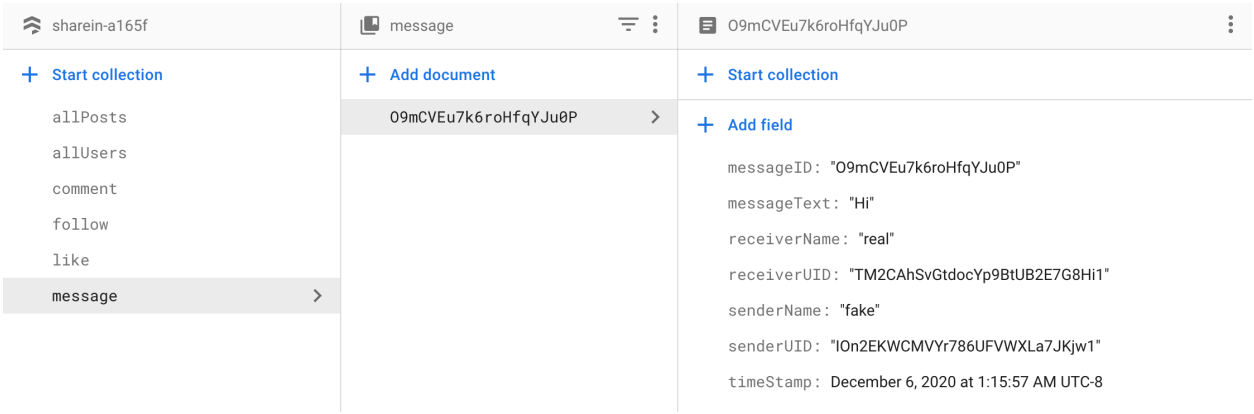

sharein-a165f
allPosts
qfZnyFs2zblX9kkh9e8z
Start collection
Add document
Start collection
allPosts
allUsers
comment
follow
like
message
0u/JRSGoH6ae3tb1k1yL
5uG1OXNMLHYrMKFPwQ6B
92zD5t3hV1lvdSH5wmO5
ERKBkMUvlTmla3oqkTDb
HX07Fvb6ykMUtzn4cGp9
NEGI2RPLELLVSXffR6on
USl3O74L17uKt4CGTQ3G
WaEBhNkN0ycACzyGh1NE
YispLlSoYaGkgZUlgak8
Zny1Y6WjjydLnOhdqL5Q
k8zoMzyTSge16dh10Q6h
lpEDnQqUcabYfzedqxXY
pkwL8paNyyzJPbE2fas3
qfZnyFs2zblX9kkh9e8z
Add field
address: ">"
musicRawID: 2131755008
name: "real"
ownerUid: "TM2CAhSvGtdocYp9BtUB2E7G8Hi1"
pictureUUIDs
postID: "qfZnyFs2zblX9kkh9e8z"
text: ""
timeStamp: December 5, 2020 at 7:29:40 PM UTC-8
title: "Hahah"

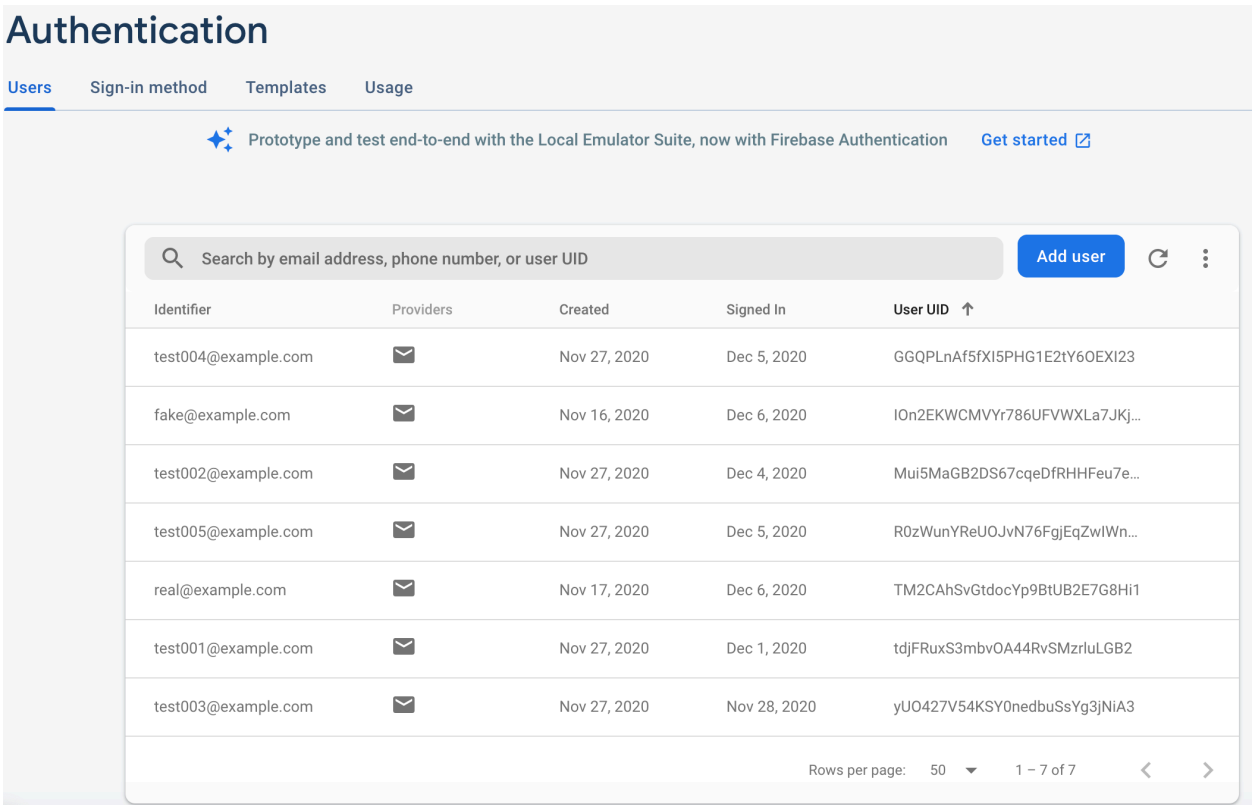

sharein-a165f
allUsers
TM2CAhSvGtdocYp9BtUB2E7G8Hi1
Start collection
Add document
Start collection
allPosts
allUsers
comment
follow
like
message
IOn2EKWCMVYr786UFVWXLa7JKjw1
TM2CAhSvGtdocYp9BtUB2E7G8Hi1
Add field
email: "real@example.com"
likedPosts
name: "real"
postsList
(array)
0
"qfZnyFs2zblX9kkh9e8z"
profilePhotoUUID: ""
uid: "TM2CAhSvGtdocYp9BtUB2E7G8Hi1"
userNameSet: true


sharein-a165f
Start collection
allPosts
allUsers
comment
follow
like
message
comment
Add document
QUwtYOLojSnd9Avkvxhy
yX5hLyNDG9e5dNuh7EQP
QUwtYOLojSnd9Avkvxhy
Start collection
Add field
commentID: "QUwtYOLojSnd9Avkvxhy"
postID: "USl3O74L17uKt4CGTQ3G"
text: "Great"
timeStamp: December 5, 2020 at 5:48:24 PM UTC-8
userName: "fake"
userUID: "IOn2EKWCMVYr786UFVWXLa7JKjw1"


sharein-a165f
Start collection
allPosts
allUsers
comment
follow
like
message
follow
Add document
Op9wcFA3VB46XZEVeJ81
mZAZVeiBbfJzURnZsEVu
Op9wcFA3VB46XZEVeJ81
Start collection
Add field
followID: "Op9wcFA3VB46XZEVeJ81"
follower: "IOn2EKWCMVYr786UFVWXLa7JKjw1"
following: "TM2CAhSvGtdocYp9BtUB2E7G8Hi1"


sharein-a165f
Start collection
allPosts
allUsers
comment
follow
like
message
like
Add document
05TRN3NDOmgDKpvJKCsy
CjGiW1EmTrAaI9CY6Mvi
SkhHxfyB8noRBcyajqm4
nfbmDdPMhrbUi5D8H6ZM
sxSJaRa0X8TBhiaXwscd
05TRN3NDOmgDKpvJKCsy
Start collection
Add field
likeID: "05TRN3NDOmgDKpvJKCsy"
postID: "5uG1OXNMLHYrMKFPwQ6B"
userUID: "IOn2EKWCMVYr786UFVWXLa7JKjw1"





(all these accounts have pw: 123456)

XIV. Code Count

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Kotlin                          36            868            225           3500
XML                             63            242             14           1970
-------------------------------------------------------------------------------
SUM:                            99           1110            239           5470
-------------------------------------------------------------------------------
```