My Knowledge Buddy

My Knowledge Buddy is an educational application made in Android to help users acquire and strengthen their knowledge on different topics of interest. Within the application, users can search and browse courses, take lessons, track progress, test knowledge, learn from their mistakes and find out how other learners are doing all at the same time.

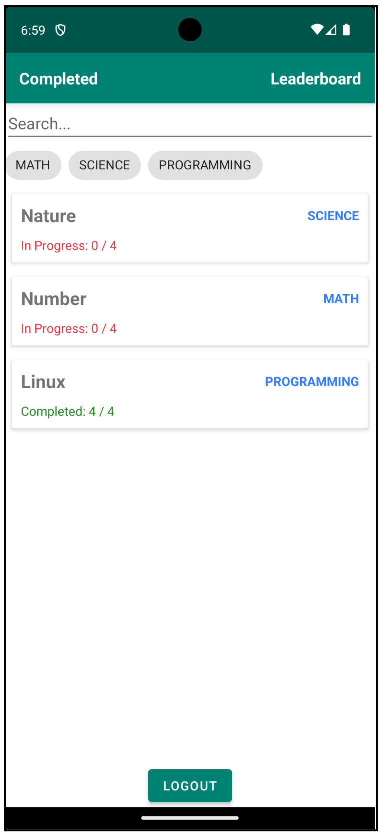
- Report your team members

Name: Nham Quoc Hung

EID: hqn278

- Include a screenshot of your app, hopefully one that illustrates key functionality.

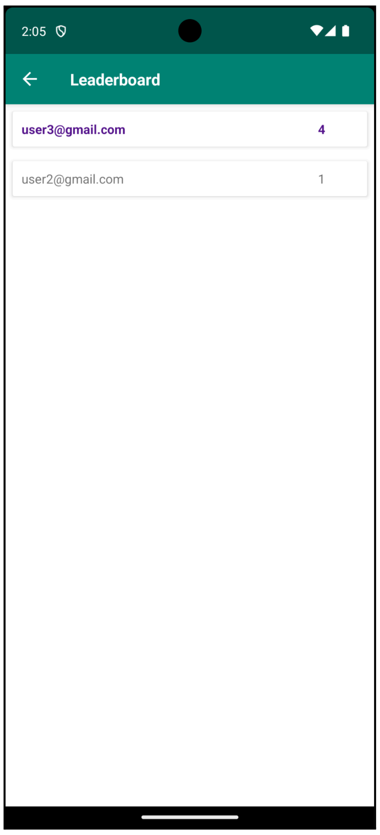
Below are some screenshots that showcase the core functionality of the application



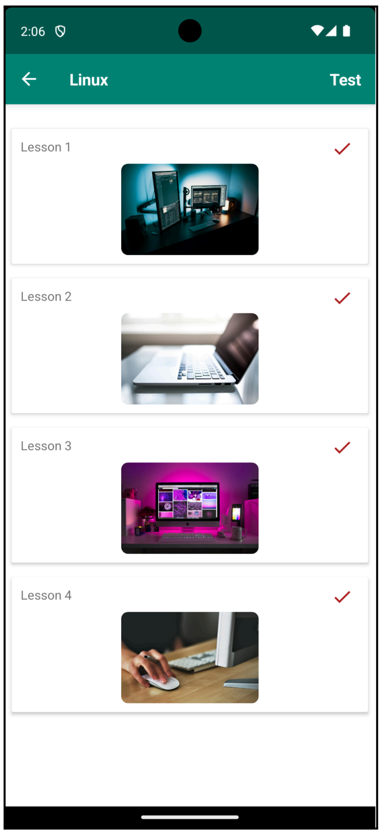
Nature SCIENCE
In Progress: 0 / 4

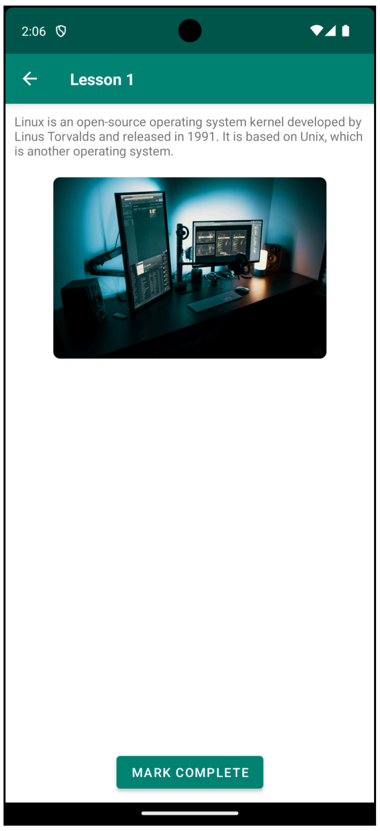
Number MATH
In Progress: 0 / 4

Linux PROGRAMMING
Completed: 4 / 4

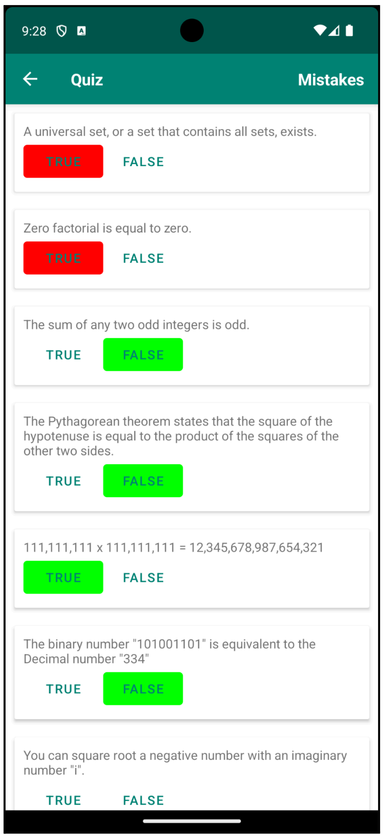


user3@gmail.com	4
user2@gmail.com	1





Linux is an open-source operating system kernel developed by Linus Torvalds and released in 1991. It is based on Unix, which is another operating system.



A universal set, or a set that contains all sets, exists.
TRUE FALSE

Zero factorial is equal to zero.
TRUE FALSE

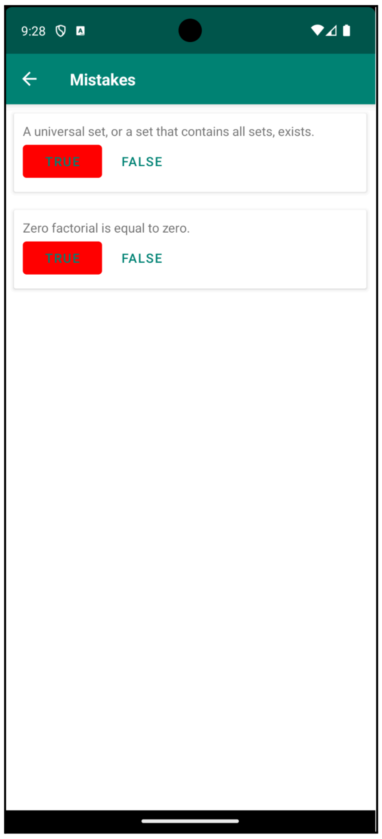
The sum of any two odd integers is odd.
TRUE FALSE

The Pythagorean theorem states that the square of the hypotenuse is equal to the product of the squares of the other two sides.
TRUE FALSE

111,111,111 x 111,111,111 = 12,345,678,987,654,321
TRUE FALSE

The binary number "101001101" is equivalent to the Decimal number "334"
TRUE FALSE

You can square root a negative number with an imaginary number "i".
TRUE FALSE



A universal set, or a set that contains all sets, exists.
TRUE FALSE

Zero factorial is equal to zero.
TRUE FALSE

- List the APIs your app uses, for example Google maps. Also let me know if you use certain Android features extensively like animation or you have custom controllers (navigation drawer, action bar, etc.).

The application mainly uses Firebase Authentication, Firestore Database, Firebase Storage and the Trivia API to achieve its desired functionality.

- List all third party libraries you use, and what they do for your app (each in their own paragraph). Describe briefly what was good and/or challenging about using them.

In the application, I used a number of third party libraries listed below to achieve different functionalities.

Glide: I used Glide to display the image for each Lesson from Firebase storage. The good thing about Glide is its ease of use across different parts of the application once it has been set up properly. However, it was slightly challenging to use at the start as it required a few different imports such as *annotationProcessor*, *compiler* as well as the *kotlin-kapt* plugin which I had to check carefully from the documentation. Also, Glide's integration with Firebase storage documentation seems slightly outdated, and I had to switch from *AppGlide* to *Glide* for it to work. Also, I faced another issue with its caching behaviour, which resulted in the old image still getting loaded when I used the same path but changed the image content from Firebase Storage. To prevent this behaviour, I had to add `DiskCacheStrategy.NONE` to its configuration. Another issue which I faced with Glide was that it would fail to load some images randomly upon using *GridLayoutManager* with 2 columns for my *Adapter*. In the end, I resorted to using *LinearLayoutManager* to properly display all the Lesson images.

RecyclerView: I used RecyclerView extensively to display different lists such as the Full Course List, Filtered Course List, Completed Course List, Lesson List, Quiz List, Mistakes List. Overall, RecyclerView is an important library and it helps to enable core functionalities of the application.

ConstraintLayout: ConstraintLayout was used to ensure responsive and correct design for the application. Many times, it is more straightforward to use

ConstraintLayout than LinearLayout as it allows me to set up a more complex layout easily. Some examples would be to constrain a button to the bottom of the parent layout without having any View on top of it.

Material Design: I used Material Design for some UI components such as *ChipGroup* and *Chip* in order to implement selection and filtering capability for my Courses browsing page. It helps to add some elegant UI/UX to the application and makes it more interesting to use.

ViewModel: ViewModel is one of the most important libraries to support the entire usage of the application. As the application consists of many Fragments which are navigated back and forth, I need the ViewModel to act as a communication layer where different Fragments can observe live and access shared data as well as call different business logics. I used the ViewModel extensively to store LiveData which tracks real-time updates from all Firestore collections, application data which is generated from different users' interactions such as changing the title upon navigation to different Fragments, doing searching and filtering of the Courses, storing questions from Trivia API, tracking completed Courses, etc. At the same time, the ViewModel also exposes functionalities from Firestore database and Firebase Storage. The challenge with using the ViewModel was mainly to get used to using it and deciding which data needed to be stored there to support the application logic.

LiveData: MutableLiveData allows application data as well as data from Firestore database to be tracked and updated in real-time so that different Fragments can reflect their data correctly to users. At the same time, MediatorLiveData allows me to implement data that reacts to changes in the source data such as getting completed Courses from the full Courses list, getting filtered Courses from any user's searching or filtering action.

Fragment: Fragment allowed me to implement different pages with separate layouts and logics. This helps to break down the UI into manageable chunks with separation of concerns.

Navigation: Navigation was used to tie the application flow together through supporting user navigation across different Fragments. I used ToorBar to support navigating back and forth between Fragments, and also to pass arguments between them. However, the challenge was to ensure valid navigation in order not to crash the application.

Retrofit: Retrofit was used to perform API calls such as HTTP GET requests while also passing in request parameters. At the same time, I also added some HttpLoggingInterceptor to debug each request. The main challenge was to set up the correct response type as well as build the correct Retrofit instance to create the correct API interface object. Once that is correct, calling the API was straightforward.

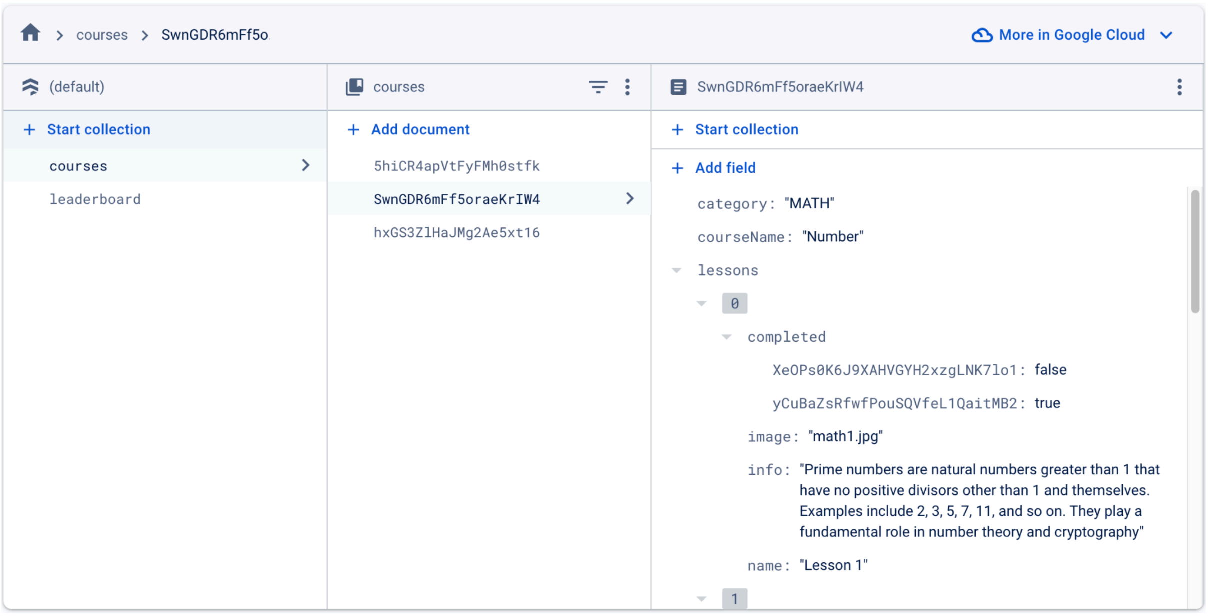
Coroutines: I used Coroutines to make asynchronous API calls. I think this is an important functionality to maintain the responsiveness of the application with non-blocking calls. The main challenge with using Coroutines was to understand how it works and different types of Dispatchers' Thread so as to pass the result back correctly to the main Thread.

SwipeRefreshLayout: I used SwipeRefreshLayout to refresh the Quiz Questions whenever a user swipes the Quiz page. It was not difficult to implement, but one issue I faced was the rate limit on Trivia API which would return an error result if the user refreshes too frequently. To resolve this, I displayed an empty page if the user refresh failed, and only displayed results if it succeeded.

- List all third party services you use, and what they do for your app (each in their own paragraph). Describe briefly what was good and/or challenging about using them.

I mainly used Firebase and the Trivia API as my third party services. In particular, I used the following services:

Firestore Database: I used Firestore Database to store data, provide real-time updates as well as support concurrent access. As different users share the same Courses and Lessons content, I stored all Lessons within each Course and all Courses within a single collection, which acts as a mutable shared state for all users. Within each Lesson, I stored the completion statuses for all users as a mapping between User ID to completion status.


courses > SwnGDR6mFf5o

More in Google Cloud

(default)

+ Start collection

courses

leaderboard

courses

+ Add document

5hiCR4apVtFyFMh0stfk

SwnGDR6mFf5oraeKrIW4

hxGS3Z1HaJMg2Ae5xt16

SwnGDR6mFf5oraeKrIW4

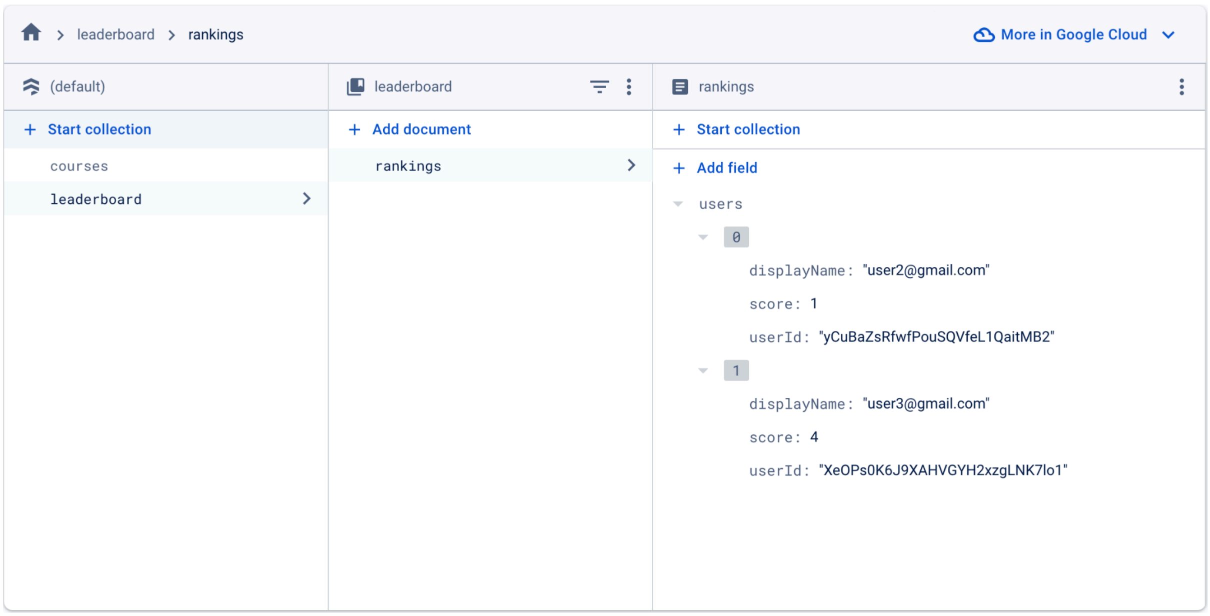
+ Start collection

+ Add field

```
category: "MATH"
courseName: "Number"
lessons
  0
    completed
      XeOPs0K6J9XAHVGYH2xzgLNK7lo1: false
      yCuBaZsRfwfPouSQVfeL1QaitMB2: true
    image: "math1.jpg"
    info: "Prime numbers are natural numbers greater than 1 that
          have no positive divisors other than 1 and themselves.
          Examples include 2, 3, 5, 7, 11, and so on. They play a
          fundamental role in number theory and cryptography"
    name: "Lesson 1"
  1
```

As shown, the *courses* collection has 3 courses with unique IDs. For a course such as “Number”, I store the Course details under *category <String>* and *courseName <String>* fields while its Lessons under the *lessons* field which is an *<Array<Object>>*. For each Lesson object, I keep track of the completion status of each user as a mapping between User ID to completion status under *completed*<Map<String, Boolean>>, as well as other Lesson details such as *name<String>, info<String>* and *image<String>*. The *image* is a path reference to a Firebase Storage image file which is used to display the Lesson image.

Besides, to implement the Leaderboard, I use another *leaderboard* collection which has a single *rankings* document. This document has a single *users* field which is an *<Array<Object>>* to store each user’s information for ranking. In particular, each User consists of a *displayName<String>*, *score<Int>* and *userId<String>*. The *userId* field is to check whether a logging-in user is a new or existing user, so as to determine whether we need to add new data for that user. The *score* field is for sorting to rank all users, while *displayName* is for showing a human-readable display name for each user.

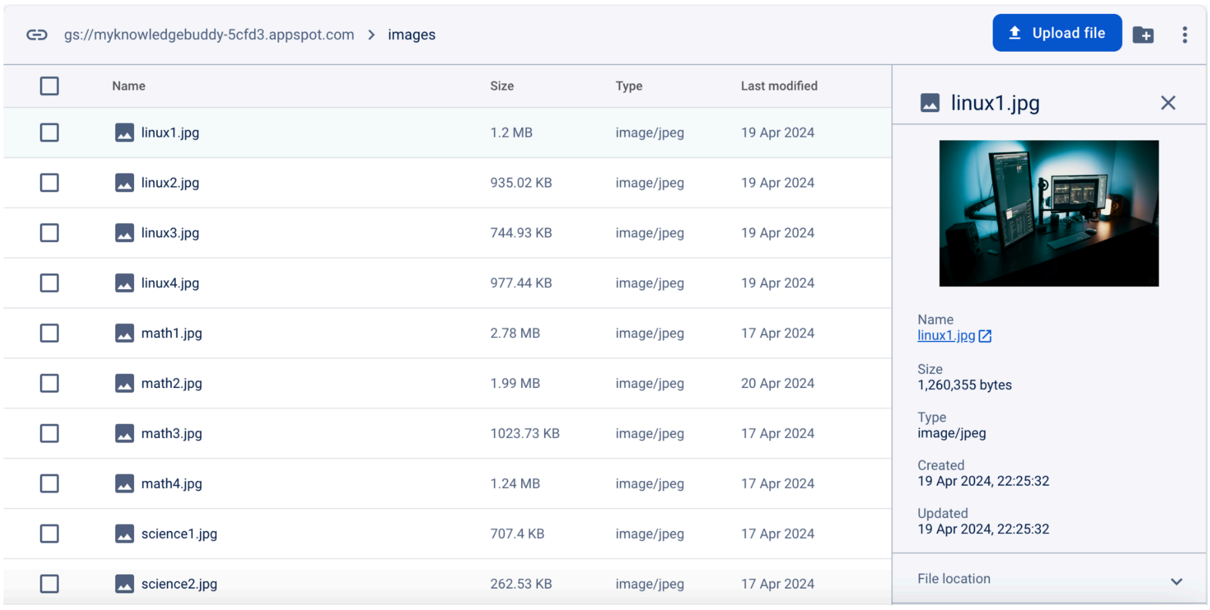


```
users
  0
    displayName: "user2@gmail.com"
    score: 1
    userId: "yCuBaZsRfwfPouSQVfeL1QaitMB2"
  1
    displayName: "user3@gmail.com"
    score: 4
    userId: "XeOPs0K6J9XAHVGYH2xzgLNK7lo1"
```

The challenging part about using Firestore Database was the schema design as well as actual implementation. Since the schema is quite nested, it was quite challenging to retrieve and map it to local objects as well as perform field updates. Specifying the field types correctly within the application was also an obstacle, as I had to step-by-step break down the Snapshot data and cast any *Any* type to the correct type and handle NULL fields to prevent the application from crashing due to NullPointerException.

Firebase Authentication: I use Firebase Authentication for user login and logout. It also helps me keep track of the current User ID to update the corresponding user's data correctly and concurrently without affecting other users. To track an user's login activities, I used *FirebaseAuth.AuthStateListener*, which can track changes to the *Authentication State*. As such, when a user login, I first check if the User ID exists within the *leaderboard* collection to determine whether this is a new or existing user. If a new user is detected, a new User Object is added to *leaderboard - rankings - users*, and a new mapping between User ID and completion status is added to all existing *courses - Course ID - lessons - completed*. This ensures a new user's information is added while no change is made for an existing user's login.

Firebase Storage: Every Lesson stores a reference to a Firebase Storage image under the *images* directory with the corresponding name. This image will be fetched and displayed by Glide when the user views all Lessons associated with a Course.


gs://myknowledgebuddy-5cfd3.appspot.com > images

Name	Size	Type	Last modified
linux1.jpg	1.2 MB	image/jpeg	19 Apr 2024
linux2.jpg	935.02 KB	image/jpeg	19 Apr 2024
linux3.jpg	744.93 KB	image/jpeg	19 Apr 2024
linux4.jpg	977.44 KB	image/jpeg	19 Apr 2024
math1.jpg	2.78 MB	image/jpeg	17 Apr 2024
math2.jpg	1.99 MB	image/jpeg	20 Apr 2024
math3.jpg	1023.73 KB	image/jpeg	17 Apr 2024
math4.jpg	1.24 MB	image/jpeg	17 Apr 2024
science1.jpg	707.4 KB	image/jpeg	17 Apr 2024
science2.jpg	262.53 KB	image/jpeg	17 Apr 2024

linux1.jpg

Name
linux1.jpg

Size
1,260,355 bytes

Type
image/jpeg

Created
19 Apr 2024, 22:25:32

Updated
19 Apr 2024, 22:25:32

File location

Trivia API: Implementing the Trivia API was not particularly difficult, as we have had one Flipped Classroom working on this API so I was more familiar with it. However, one of the functionalities was to track and show Mistakes made by the user, so I had to additionally track the user's answer choice for each question to show where they got it wrong and preserve this information even if the user navigates to another Fragment. To implement this, I added another field to each Question object to track user's selection, and display it correspondingly whenever the user enters the same Fragment, until the user refreshes it.

- List any component substantially generated by AI. Describe your prompts and briefly summarise what was good and/or challenging about the process.

Since I mostly used existing ideas and knowledge gained from previous Flipped Classrooms, I did not have to rely on AI substantially for this Project. For some new functionalities such as listening to Firestore database's real-time updates or performing concurrent and transactional updates to the database, I cross-referenced Firebase's documentation with ChatGPT to understand the behaviour. One idea which I found interesting was that for any transactional update, we will have to read the data first before any subsequent updates, and a failure at any step will invalidate the entire transaction. As a result, I mainly used ChatGPT to generate some demo Lesson's content under the *info<String>* field to display in the Lesson page.

- Discuss anything noteworthy about your UI/UX/display code.

My UI/UX/display code mainly uses XML for layout specification. Perhaps the most interesting layout is from the main Courses browsing page where there's a Search Bar, a Chip Group and a RecyclerView list of Courses. Any searched keyword will filter the Courses with text highlighting, and any clicked Chip will filter the Courses with the corresponding *category<String>* field.

- Discuss anything noteworthy about your back end or processing logic.

The most noteworthy thing about my processing logic is the new user's data creation flow as I have mentioned upon a user's login, the real-time updates from *courses* and *leaderboard* collections upon any changes as well as support for concurrent access to these collections as shared mutable states. For the last point, I used Firebase's transaction feature to ensure safe concurrent updates across multiple users.

- Discuss the most important or interesting thing you learned doing your project.

The most interesting thing I learned was a deeper understanding of how different concepts taught in class such as Layout, LiveData, RecyclerView, ViewModel, Navigation, API and Firebase came together to support this final Project. From doing a majority of the project setup and implementation myself, I understood how to properly structure different components of an Android application to implement the overall application logic. I was most impressed with the ViewModel in particular on how it helps to ensure a unidirectional and live data flow to different Fragments, as well as the smooth integration between MutableLiveData and Firestore's real-time updates. Another part that impressed me was seeing Coroutines in action for asynchronous updates of the LiveData as well.

- Discuss the most difficult challenge you overcame and/or your most interesting debugging story.

The most difficult challenge was the Firestore schema design as well as its real-time and transactional features to support the whole application logic. I was mostly concerned with this part since it would determine whether my proposed functionalities were achievable at all. I mainly did brainstorming as well as trial and error on this part and faced many bugs. Perhaps my most interesting bug was related to detecting whether a logged-in user is new or existing to create new data for that user. Initially, I compared if the user's *lastSignInTimestamp* and *creationTimestamp* were equal to decide that user is new, assuming that upon the application restart, these timestamps would surely differ. However, they only differed if the user explicitly logged out and not upon the application restart, causing me to falsely detect an existing user as new and add or overwrite their data unknowingly. Upon many attempts to resolve this such as changing from Quick to Cold Boot, overriding *onStop()* and *onDestroy()* or disabling the Device Manager's *Save quick-boot state on exit*, I used a more reliable approach which is to check the *leaderboard* collection for any existing User ID to determine instead.

- If necessary, briefly tell us how to build and run your project. Include details about how to set up back end services (if you use them). In the common case, we will rely on your demo, but just in case we have an issue, we'd like some tips.

- In your writeup, clearly identify the location of your repository. Submit your final code to this new project. For masters students, do the same, but use the masters class organisation. You can create your own private repo.

Repository location: https://github.com/ut-msco-s24/MyKnowledgeBuddy

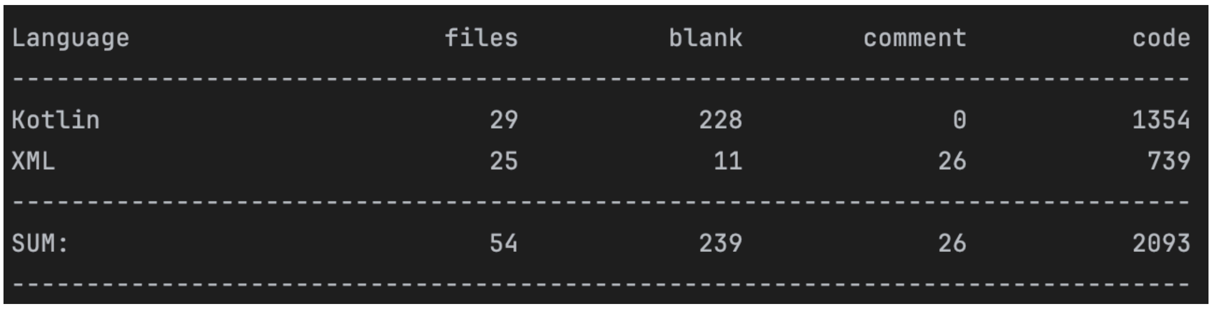
- For the masters class, you need a video demo. Please include a link to your video demo in your writeup. The video should be publicly accessible to anyone with the link.

Demo video:
https://drive.google.com/drive/folders/1AXRldiMxh9spf8Of6xCdiqeMFFpBWojl?usp=sharing

- If you use firebase, include a printout (or screenshot) of your database schema. The schema is basically the structure and the type information. So print out the keys and values and the types for any important data you have in the database. Here is a simple example.

Printout is shown in some sections above

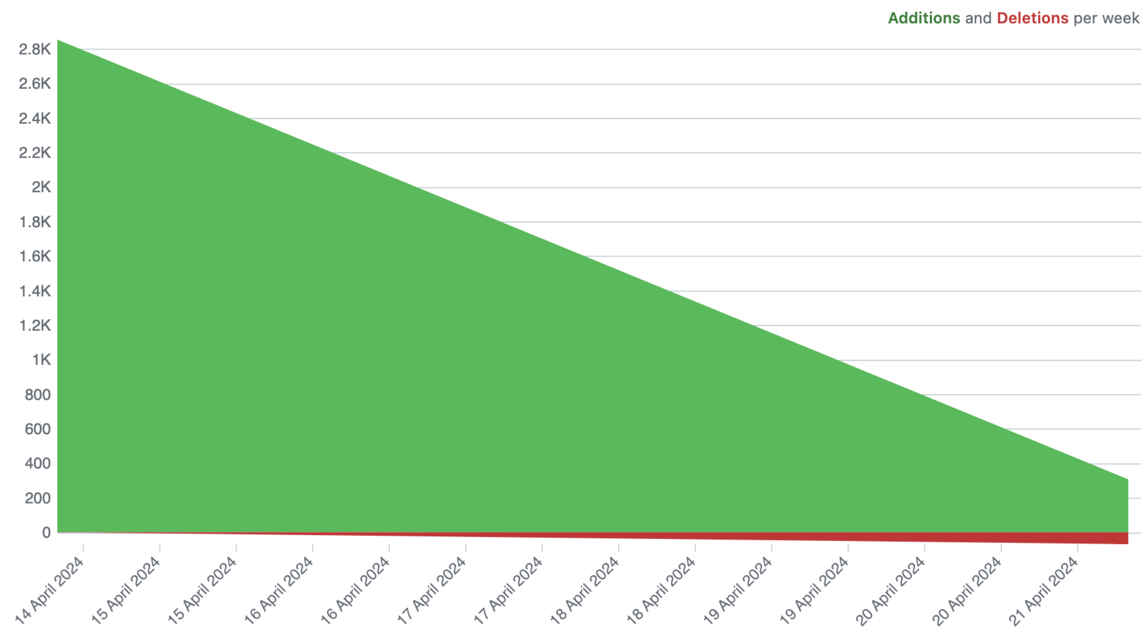
- Report the count of lines of the code in your project



```
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Kotlin                          29            228              0           1354
XML                             25             11             26            739
-------------------------------------------------------------------------------
SUM:                            54            239             26           2093
-------------------------------------------------------------------------------
```

- Code frequency graph

Code frequency over the history of **ut-msco-s24/MyKnowledgeBuddy**


Additions and Deletions per week
2.8K
2.6K
2.4K
2.2K
2K
1.8K
1.6K
1.4K
1.2K
1K
14 April 2024
15 April 2024
15 April 2024
16 April 2024
16 April 2024
17 April 2024
17 April 2024
18 April 2024
18 April 2024
19 April 2024
19 April 2024
20 April 2024
20 April 2024
21 April 2024