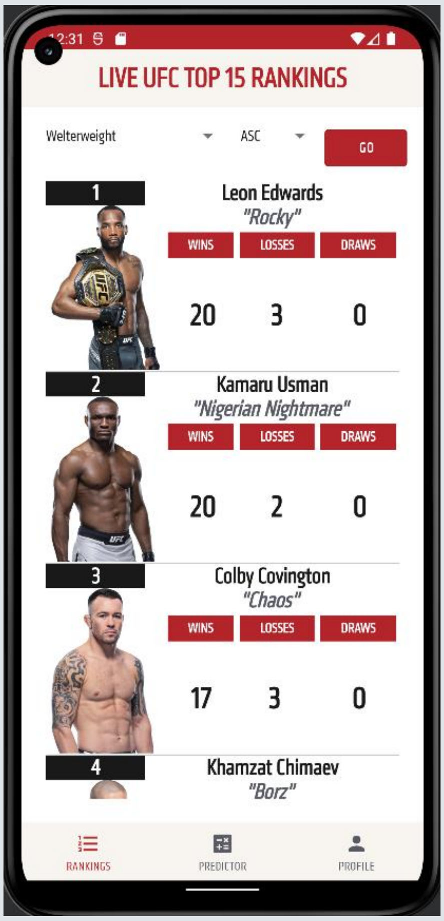
ANDROID PROGRAMMING (WB) PROJECT WRITEUP: UFC MMA PREDICTOR

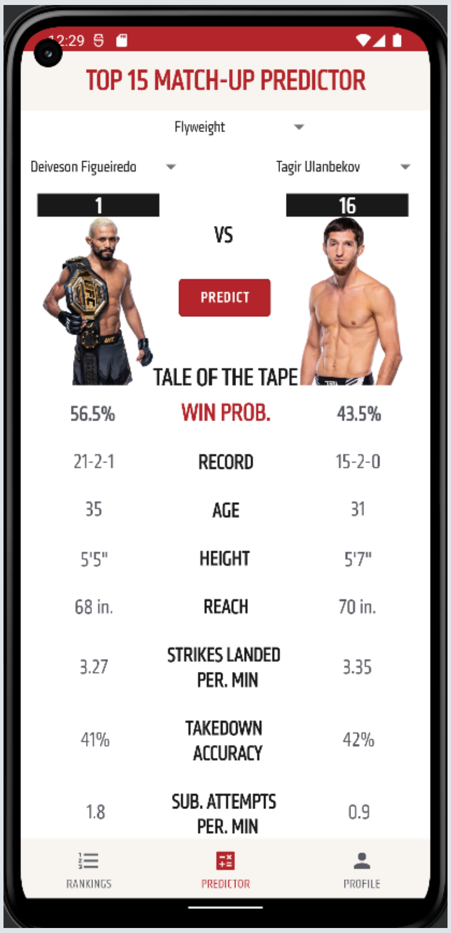
App Title: UFC MMA Predictor
Full Name: Jason Chan Jin An
UID: jjc5439
Email: jasonchan@utexas.edu

The World's First Ever
UFC MMA Predictor Mobile App AI Powered
Predictions and Analytics
at the Tip of your Fingers!


LIVE UFC TOP 15 RANKINGS

Welterweight ▾ ASC ▾ GO

Rank	Fighter	WINS	LOSSES	DRAWS
1	Leon Edwards "Rocky"	20	3	0
2	Kamaru Usman "Nigerian Nightmare"	20	2	0
3	Colby Covington "Chaos"	17	3	0
4	Khamzat Chimaev "Borz"			


TOP 15 MATCH-UP PREDICTOR

Flyweight ▾
Deiveson Figueiredo ▾ Tagir Ulanbekov ▾

1 VS 16

PREDICT

TALE OF THE TAPE

Deiveson Figueiredo		Tagir Ulanbekov
56.5%	WIN PROB.	43.5%
21-2-1	RECORD	15-2-0
35	AGE	31
5'5"	HEIGHT	5'7"
68 in.	REACH	70 in.
3.27	STRIKES LANDED PER. MIN	3.35
41%	TAKEDOWN ACCURACY	42%
1.8	SUB. ATTEMPTS PER. MIN	0.9

UFC MMA PREDICTOR

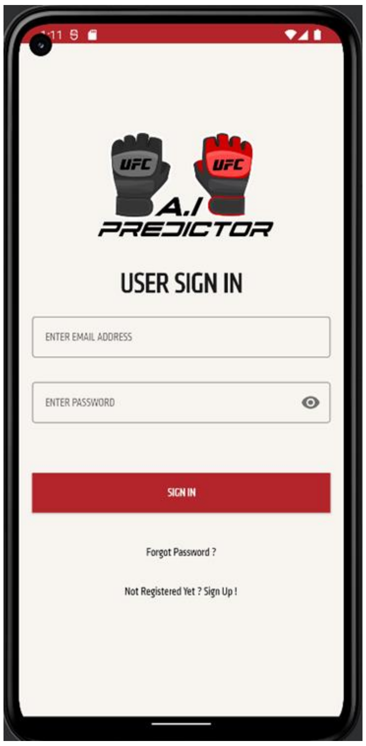
The University of Texas at Austin
Department of Computer Science
College of Natural Sciences

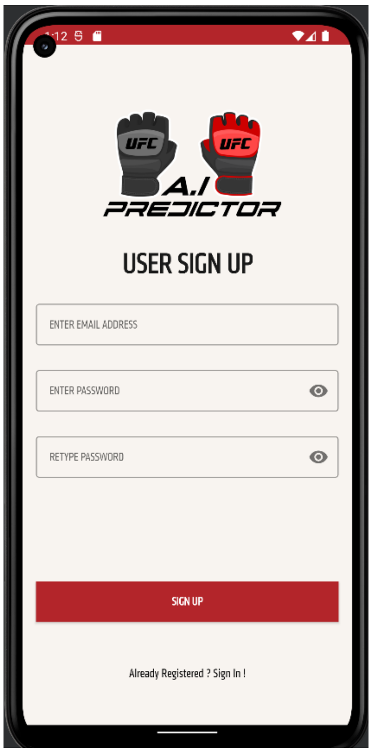
1. APP SCREENSHOTS

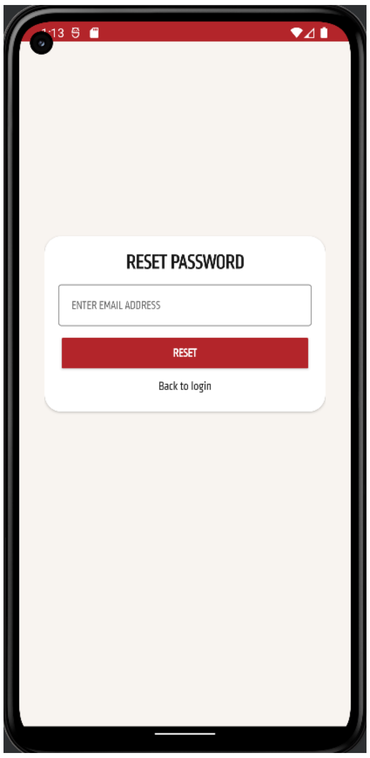
Below are the screenshots of the application categorized into authentication, fighter rankings, predictor, and profile alongside their key functionalities.

Authentication

Once the app is launched, the user can sign up via email, sign in, and reset password. Firebase authentication API with custom page was implemented.



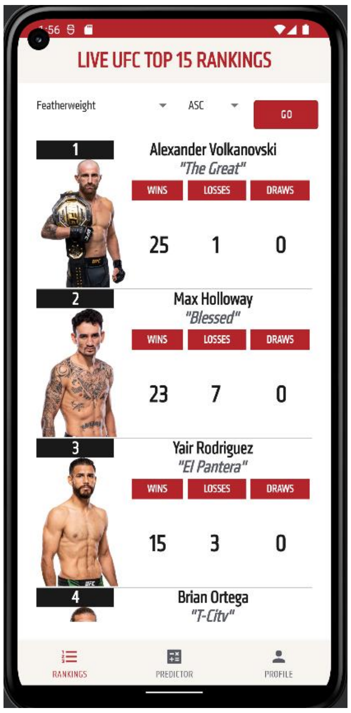




Fighter Rankings

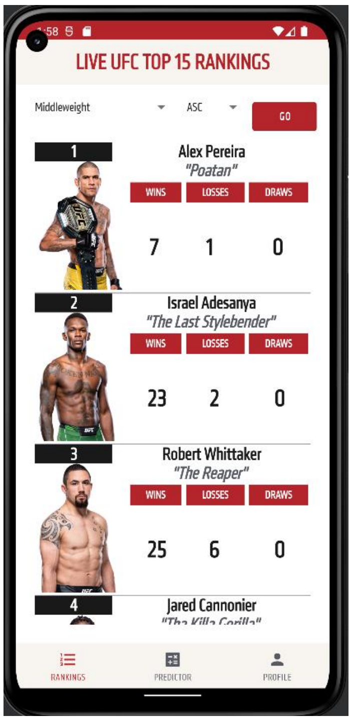
Once the user successfully logs in, the user will be presented with the first fragment, which is the fighter rankings. Here, the user can select the weight class of interest alongside presenting in either ascending or descending and click “GO”. The dropdown list was implemented using spinners and the scrolling view was done using RecyclerView. Noted that the rankings data and images are live as the data is obtained from daily web scrapping of the UFC rankings website and accessed using a container service API (will be explained in later section).

1. APP SCREENSHOTS


LIVE UFC TOP 15 RANKINGS

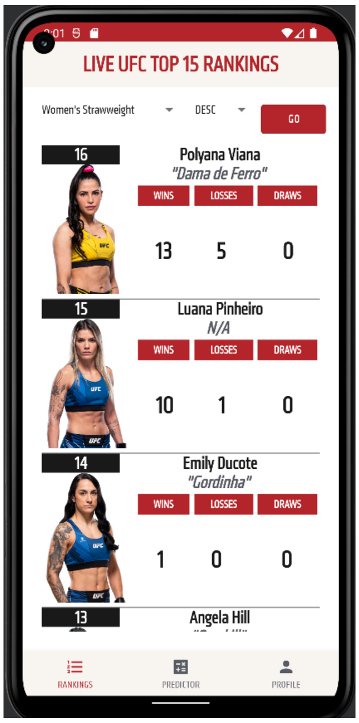
Featherweight ASC GO

Rank	Fighter	WINS	LOSSES	DRAWS
1	Alexander Volkanovski "The Great"	25	1	0
2	Max Holloway "Blessed"	23	7	0
3	Yair Rodriguez "El Pantera"	15	3	0
4	Brian Ortega "T-City"			


LIVE UFC TOP 15 RANKINGS

Middleweight ASC GO

Rank	Fighter	WINS	LOSSES	DRAWS
1	Alex Pereira "Poatan"	7	1	0
2	Israel Adesanya "The Last Stylebender"	23	2	0
3	Robert Whittaker "The Reaper"	25	6	0
4	Jared Cannonier "Tha Killa Gorilla"			

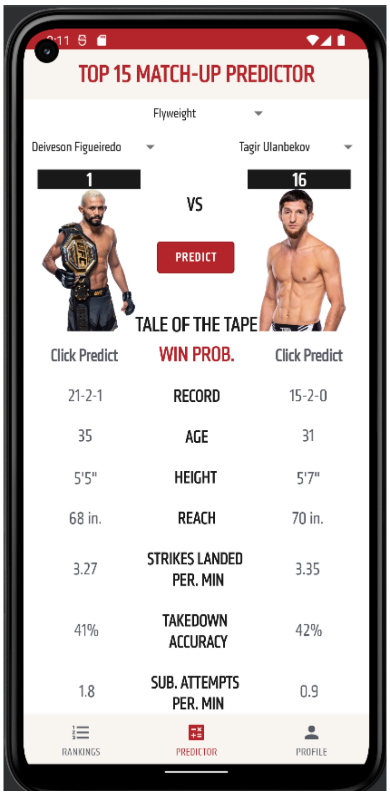

LIVE UFC TOP 15 RANKINGS

Women's Strawweight DESC GO

Rank	Fighter	WINS	LOSSES	DRAWS
16	Polyana Viana "Dama de Ferro"	13	5	0
15	Luana Pinheiro N/A	10	1	0
14	Emily Ducote "Gordinha"	1	0	0
13	Angela Hill			

Top 15 Matchup Predictor

The predictor is to make predictions on matchups in the top 15 of a selected weight class. The user would simply select the weight class and the fighters using the spinners and then click "PREDICT". The winning probability for each fighter will then be generated. The features and model used will be explained in later sections.


TOP 15 MATCH-UP PREDICTOR

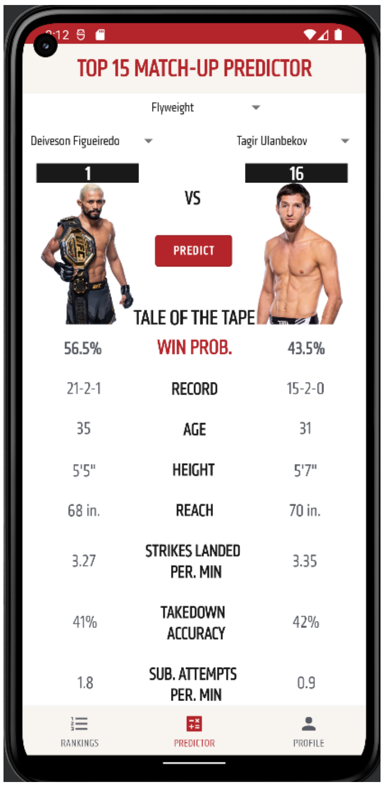
Flyweight

Deiveson Figueiredo VS Tagir Ulanbekov

1 PREDICT 16

TALE OF THE TAPE

Deiveson Figueiredo		Tagir Ulanbekov
Click Predict	WIN PROB.	Click Predict
21-2-1	RECORD	15-2-0
35	AGE	31
5'5"	HEIGHT	5'7"
68 in.	REACH	70 in.
3.27	STRIKES LANDED PER. MIN	3.35
41%	TAKEDOWN ACCURACY	42%
1.8	SUB. ATTEMPTS PER. MIN	0.9


TOP 15 MATCH-UP PREDICTOR

Flyweight

Deiveson Figueiredo VS Tagir Ulanbekov

1 PREDICT 16

TALE OF THE TAPE

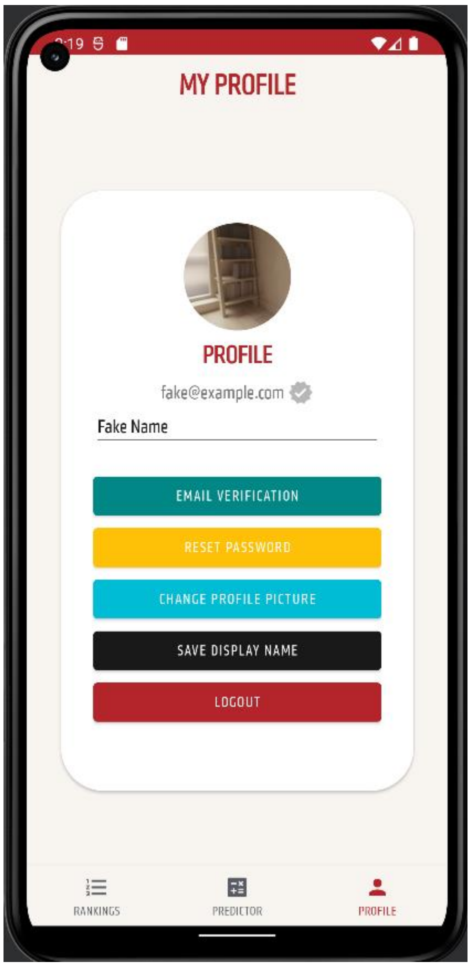
Deiveson Figueiredo		Tagir Ulanbekov
56.5%	WIN PROB.	43.5%
21-2-1	RECORD	15-2-0
35	AGE	31
5'5"	HEIGHT	5'7"
68 in.	REACH	70 in.
3.27	STRIKES LANDED PER. MIN	3.35
41%	TAKEDOWN ACCURACY	42%
1.8	SUB. ATTEMPTS PER. MIN	0.9

1. APP SCREENSHOTS

User Profile

The user profile enables the user to customize his or her own profile. The key features includes:

- set or change display name
- verify email
- reset password
- set or change profile picture
- logout



2. APIs USED

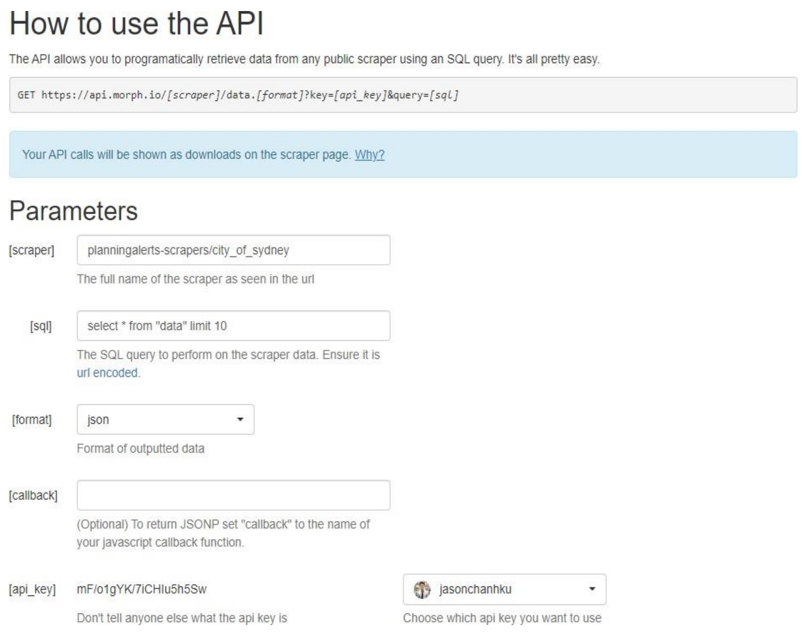
Below are the list of APIs used throughout the android application separated by it's own category and section used

API NAME	SECTION USED	DETAILS
Firebase Authentication and Storage API	• Log in screen • Sign up screen • Verify email • Reset password • Profile picture update and storage • Display name update	Firebase authentication was used to create my own custom pages to carry out user authentication related activities. Lastly for the profile picture storage, Firestore was used to save the image with the user email as the title
Glide	• Fighter rankings images • Fight predictor images	Glide was used to conveniently display the fighter images in the rankings and predictor fragments as the image link was a url that was scrapped from the UFC website
TensorflowLite	• Predicting win probabilities	TensorflowLite model was used to train on fighter stats data to predict winner of the matchup. It was relatively easy to export the .tflite file into Android Studio
morph.io	• Fighter rankings data • Fighter fight stats • Fighter images url	Morph.io was used as a container to the scrapers that were deployed and allows an API access to retrieve the scrapped data with an SQL like query parameter which returns a json.
Retrofit and Gson	• Building API interface to obtain the json data • Parsing of json for fighter rankings and fighter stats data	Retrofit and GSON was conveniently used to called the morph.io API and store in a dataclass for easy usage Retrofit and GSON was conveniently used to called the morph.io API and store in a dataclass for easy usage.

3. Third Party Services

morph.io

morph.io offers free and public open-source containerized web scrappers and takes the hassle out of web scraping. morph.io also offers regular automated scraper execution and alerting if there's a problem. Besides, morph.io also helps to store the scraped data I need for the project and making the data available for download or access through a super-simple API. The API returns in a json format and the query parameter can be done in SQL string format. It was surprisingly not too difficult to implement this due to the amount of practice with APIs and parsing json from the flipped classrooms and assignments.



How to use the API

The API allows you to programatically retrieve data from any public scraper using an SQL query. It's all pretty easy.

```
GET https://api.morph.io/[scraper]/data.[format]?key=[api_key]&query=[sql]
```

Your API calls will be shown as downloads on the scraper page. Why?

Parameters

[scraper] planningalerts-scrapers/city_of_sydney
The full name of the scraper as seen in the url

[sql] select * from "data" limit 10
The SQL query to perform on the scraper data. Ensure it is url encoded.

[format] json
Format of outputted data

[callback]
(Optional) To return JSONP set "callback" to the name of your javascript callback function.

[api_key] mF/o1gYK/7iCHIu5h5Sw
Don't tell anyone else what the api key is

jasonchanhku
Choose which api key you want to use

TensorflowLite

TensorflowLite was also simpler than expected to implement as Android Studio allows direct import of tflite models and sample code is generated right away. The more challenging part was preparing the data in the application into the format required for prediction. This include the preparation of ByteBuffers and memory to be allocated. More research was done for this part as it was the bottleneck.

5.MODEL AND FEATURES

Features

Fight Stats

The features used were mainly from the UFC Fightmetric website which includes fight stats and physical stats of each fighters. The numerical data which I scrapped were:

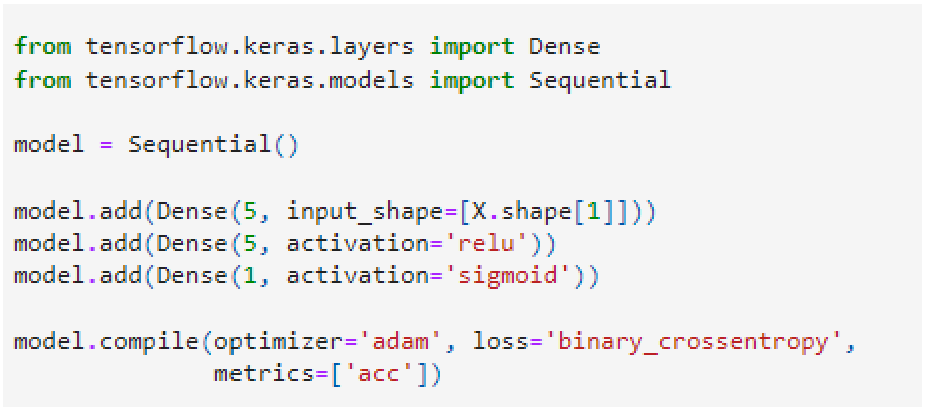
- Reach
- Height
- SLpM - Significant Strikes Landed per Minute
- Str. Acc. - Significant Striking Accuracy
- SApM - Significant Strikes Absorbed per Minute
- Str. Def. - Significant Strike Defence (the % of opponents strikes that did not land)
- TD Avg. - Average Takedowns Landed per 15 minutes
- TD Acc. - Takedown Accuracy
- TD Def. - Takedown Defense (the % of opponents TD attempts that did not land)
- Sub. Avg. - Average Submissions Attempted per 15 minutes

Feature Construction

Using the list of fights that happened, I took the difference of Fighter 1's features to subtract Fighter 2's features. The selection of Fighter 1 and Fighter 2 are at random. Hence the final dataset before being fed into the model is a dataset of difference is fight stat features of each fight that happened between 2 fighters.

Model

A relatively simple model was used as there are only around 3000 rows of data, which represents 3000 fights that took place historically. Hence 3 dense fully connected are used before the final sigmoid activation function. With the current limited features, the model is around 55% accurate with a train val split of 70:30.



```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

model = Sequential()

model.add(Dense(5, input_shape=[X.shape[1]]))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
```

6. LEARNINGS & CHALLENGES

Most Important Learning

The most important learning I acquired in this final project is the planning and execution for each of the fragments. I realized it helps a lot to take a step back and think of the trade-off between taking the path of least resistance vs satisfying the initial intended requirement for the application. By doing more thinking and planning, I got to find more efficient ways of executing what I wanted to. For instance, for the images of the fighters, I first thought of manually downloading the top 15 fighters of each weight class locally and then storing them in Firestore. However, after I took a step back I realized that this is not realistic because it would not be realistic as what if a fighter who is not a champion becomes a champion and their image would change with the belt on their shoulder? Hence, after taking a step back and evaluating my options, I realized I could leverage scheduled web scraping (using morph.io) of all data which includes the scrapping of their image URL from the UFC site. Hence, this would reflect the latest happening without additional effort from my side. This then sparked an improvement idea which I would elaborate further in the section below.

Overcoming Challenges (Debugging Story)

Before the realization that I could use morph.io web scrapping to have dynamic data, I already implemented the path of least resistance, which was having a static data in an SQLite database. This was considered as a comfort zone for me due to the experience from the flipped classroom and the simplicity of querying from the SQLite database with the rather trivial syntax. Hence, to convert all my static code using SQLite database to a dynamic morph.io API call which returns a json of a query, I needed to rewrite a significant part of my code base, primarily in the data retrieval section. This seemed very daunting at the time because I could have settled for a static dataset but my eagerness to learn and determination was overpowering. Hence, using the learning above, I took a step back to plan ahead clearly of what I needed to use and change in order to achieve the dynamic data I desired. I invested time in coming up with a clear plan of the APIs and libraries I needed for each section to replace my current pipeline. After that, I was able to execute the debugging smoothly with minimal setbacks. Looking back, I could have settled for less with the static data but I knew I have learned so much in this course and it would be a shame to settle for less. Having built an application with live updated data, I need not to worry about unfulfilled potential for this project.

Running

Just unzip the .zip file and open the folder **”android_programming_project”** with android studio and run. If failed to run, please clean project and rebuild, else feel free to reach out to me thanks.