Name: Derek Salvucci (project done solo)
EID: dts999

Final Project Writeup

App Title: MLB Pocket Guide

App Description:

A data visualization app that allows users to select favorite MLB players from the 2022 season, check out some biographical and historical performance information about individual players, and compare pitchers and hitters in key matchup indicators. By signing in to the app, users can save their favorite players across different app sessions.

APIs Used:

Excluding any other library integrations, I only used one external api "provided" by FanGraphs. I put it in quotes because the API integration was decoded by monitoring network calls on their webpage, as the endpoints and json responses are not publicly documented.

Third Party Libraries:

My project used the following libraries:

- OkHttp to handle the network calls to FanGraphs
- A variety of Android KTX extensions to enable building fragments, fetching and storing data in the background, and moving around within our app:
    - Lifecycle, LiveData, Navigation, Fragment, RecyclerView, Coroutines
- MPAndroidChart to develop the Pie Chart and Line Chart data visualizations found on the player profile page

While I am comfortable with OkHttp and the KTX Extensions I used due to our previous assignments, the MPAndroidChart library proved to be a bit of a challenge to integrate into the app. Very core functionality was easy to spin up, however navigating the documentation to get charts to look and be formatted exactly how I wanted them to be was a big source of frustration. The library is 100% Java, so from the get-go writing in Kotlin was requiring me to translate from a language I do not know at all. Additionally, when trying to customize the axes labels I found the library did not support doing something as simple as labeling the y-axis. In researching, I realized that this was by design from the creator, which created more work customizing the layout than I anticipated. For other formatting tweaks, researching the library more had me looking for functions that were ultimately deprecated, and I didn't realize until I found enough posts telling me that was the case. Overall, it was a very difficult library to work with simply because it was tricky to navigate the Java-based documentation, and translate that into Kotlin implementations.

Third Party Services

In addition to the libraries listed above, the app integrates with Google Services to provide authentication via Firebase, and persistent data storage via Firestore, of which the schema can be seen in the appendix.

These services are pretty straightforward to use, and their implementation was largely a practice in pattern-matching from the previous assignments. However, there was an additional step required for this app: in the assignment dealing with Firestore, we were not explicitly asked to fetch records only for the account signed in. In this case, I need to not only fetch, but also delete, favorite players only for the account that was signed in. Adding this where clause to the query was a little tricky, but ultimately solved.

Noteworthy UI / Display Code

My app doesn't exactly push the boundaries on what is possible for mobile UX, but building this out has given me a much greater appreciation for designers that can wireframe intuitive apps that don't require much onboarding to understand how it works.

One implementation detail that I am proud of, though not particularly challenging technically, is that all permutations of searching for a player flow through the same search fragment code. When we create an instance of the searching fragment, we feed in the location we are trying to search from in the app, and the search logic allows us to serve the correct layout, player list, and click listeners, rather than repeating ourselves and creating a different search fragment for each permutation.

Lastly, it was fun to integrate a persistent bottom navigation bar in this app as it is a pretty well utilized pattern in popular apps today, but was not a component in any of the assignments. With the tutorials available online it was easy to implement, and I felt like I was able to learn a design pattern that will be useful even after this assignment.

Noteworthy Backend Logic

By keeping one "home" activity live, and utilizing fragments for all of the interactions in the app, I was able to easily build out a cache of player information that had already been fetched from FanGraphs during that app session within one of the two view models. Doing this allows us to store the results and display repeated players without having to fetch their information over the network every time, which is nice to limit data usage and potentially speed up the app when service is bad.

Additionally, building out the app without knowing whether or not the user was signed in to a Firebase account was an easy but long process. There are so many edge cases to look out for when a user is not signed in, particularly around initializing certain variables as empty, and skipping network calls to Firestore DB, that maintaining both environments in development led to a number of bugs during testing that were squashed.

Important Things Learned

I mentioned a bigger appreciation for people that can design apps that flow well, but I learned that going from design/idea to implementation in xml layout files can be incredibly difficult and tedious, especially when trying to stand up multiple new screens at once. After building this project, I understand the motivation for design components that can be reused across many screens without having to repeat yourself rather than defining each screen within its own layout context. Modular design components that lower the effort needed to use repeatedly can save so much time and sanity.

Particular Debugging Challenge

The biggest source of frustration was during the API integration with FanGraphs. To start, there is no public documentation, so understanding what URL to make a post request to, and what shape of data would be returned was done by monitoring my network while poking around their website. Once I was pretty certain of the URL and data shape, I went to work on fitting a retrofit class for the response in order to pattern-match with our previous assignments since I knew that worked. The FanGraphs response is a bit more complex than what we had dealt with before, as there were a number of nested Arrays containing JSON Objects, and Retrofit doesn't really support nesting generic Objects within Arrays, you had to define that nesting directly. Based on the litany of Stack Overflow discussions looking for help deserializing this, I knew I was in for some trouble.

After a lot of trial and error, I was able to get *something* functional for position player API requests for FanGraphs. When I decided to input a pitcher instead of a position player in the request, FanGraphs returned a JSON that was in a different shape, breaking my Retrofit deserialization. I searched a bit to try and build a deserializer that would understand if I were receiving a position player or pitcher in return, and parse the response accordingly, but ultimately decided to scrap the Retrofit integrations in favor of a generic JSON parser. Because I knew the keys I was looking for, and the shape of the responses, it was much simpler to just read the body of the response and find the subset of fields I was looking for, though a lot less satisfying than trying to untangle this mess.

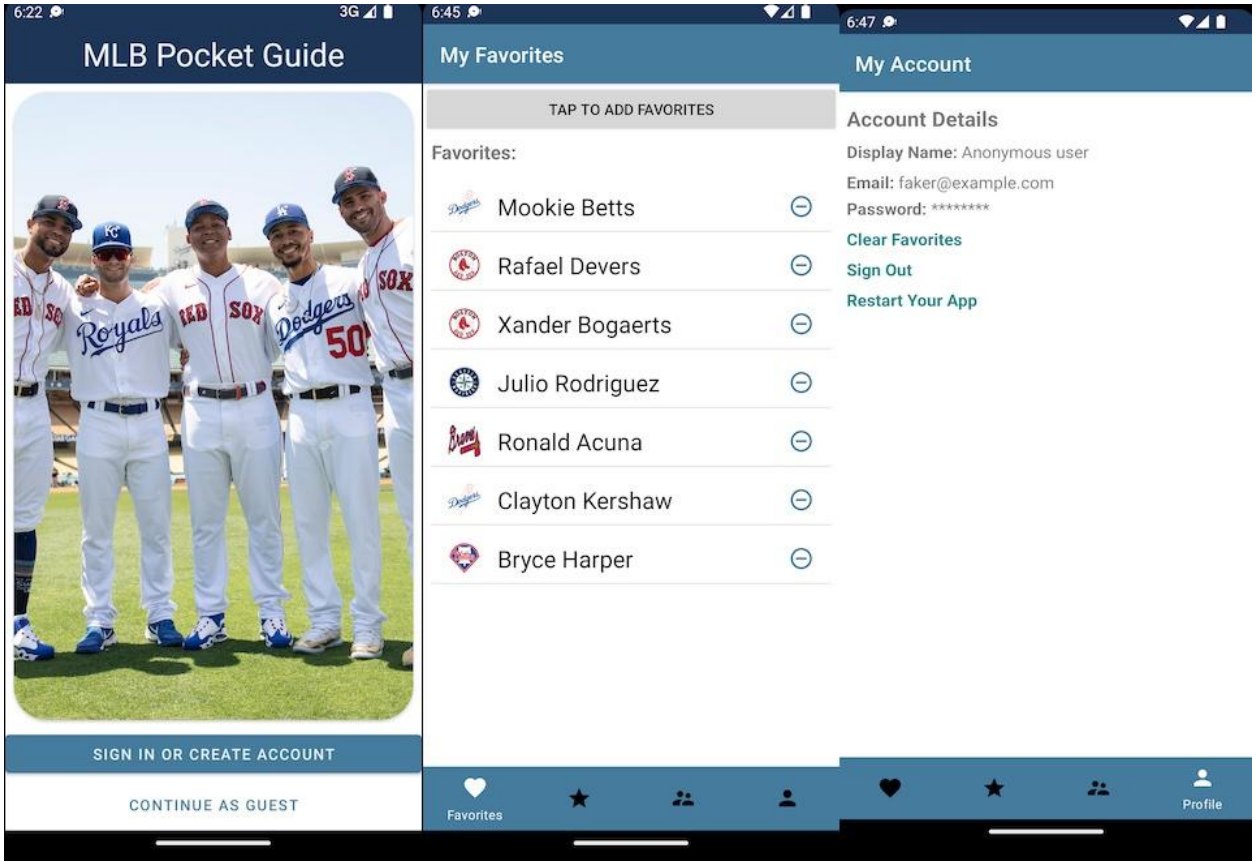
Another challenge I ran into while debugging was the "Clear Favorites" button would only delete one favorite at a time, despite calling a "forEach" on the list of favorites in my viewModel. Because the app technically threw no error message, understanding why the rest of the for loop was not respected was difficult. However, I was able to get around it by fetching the documents from Firestore rather than the viewModel, and iterating through that list to delete them based on the advice of some StackOverflow posts complaining about a similar problem, allowing me to continue.

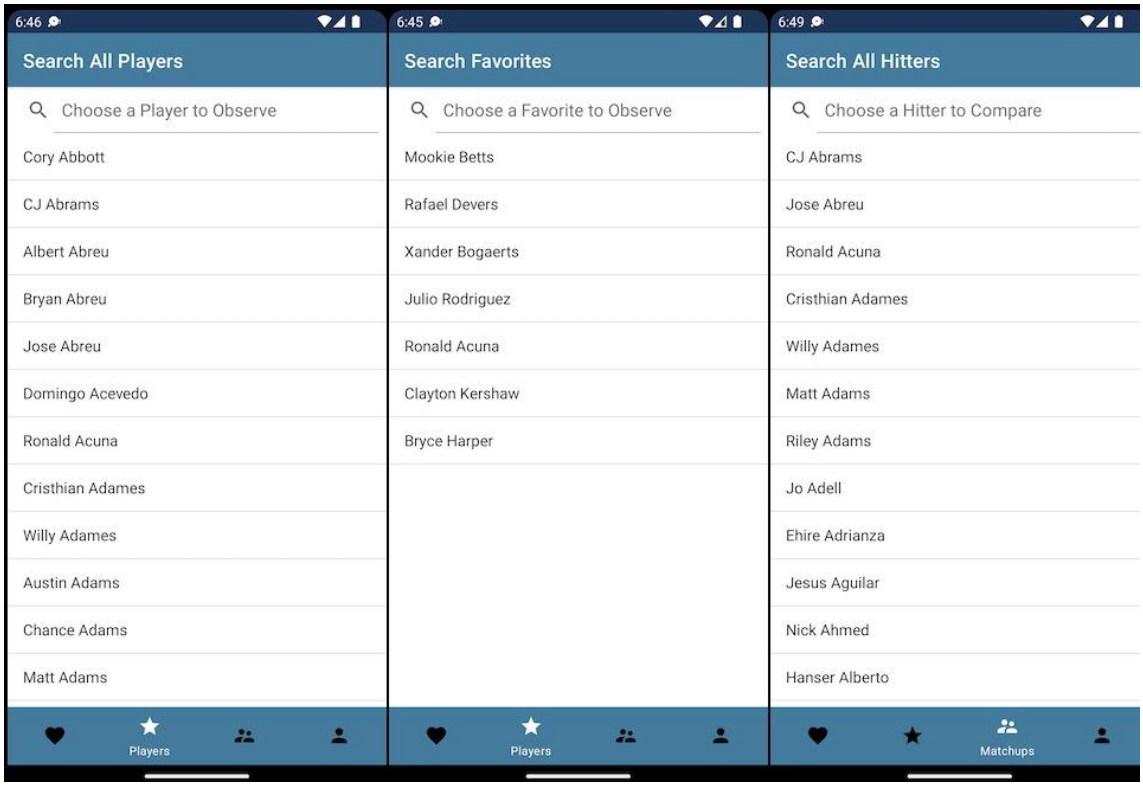
How To Build and Run

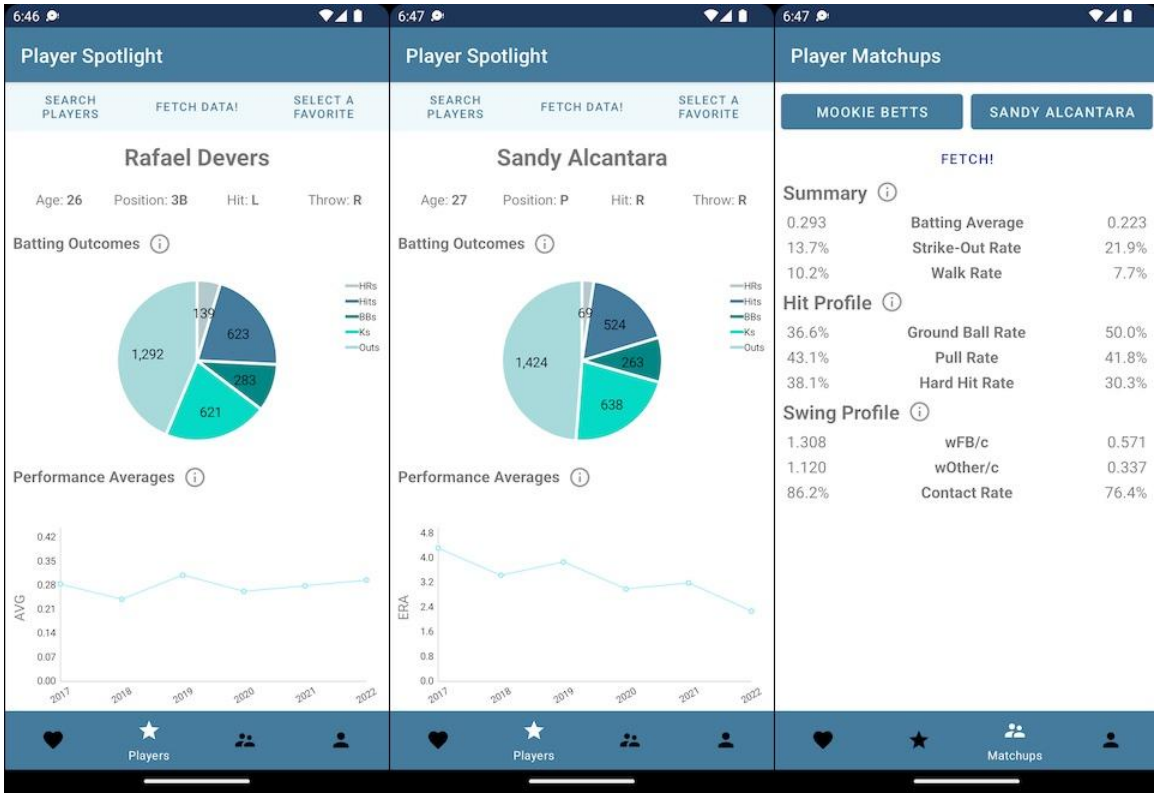
The google-services.json file is included in the .zip file, so there should be no issues with downloading that folder, unzipping, and simply building the project in Android Studio. It was developed on a Pixel 3A device, with the API set at 33, so matching to those configurations should be sufficient.

There is an option to enter the app as a guest user, the Firebase system will handle your request to create your own account if you'd like, and the email address **faker@example.com** with password **123456** has been configured to initialize with some favorites pre-populated to make that functionality easy to see as well.

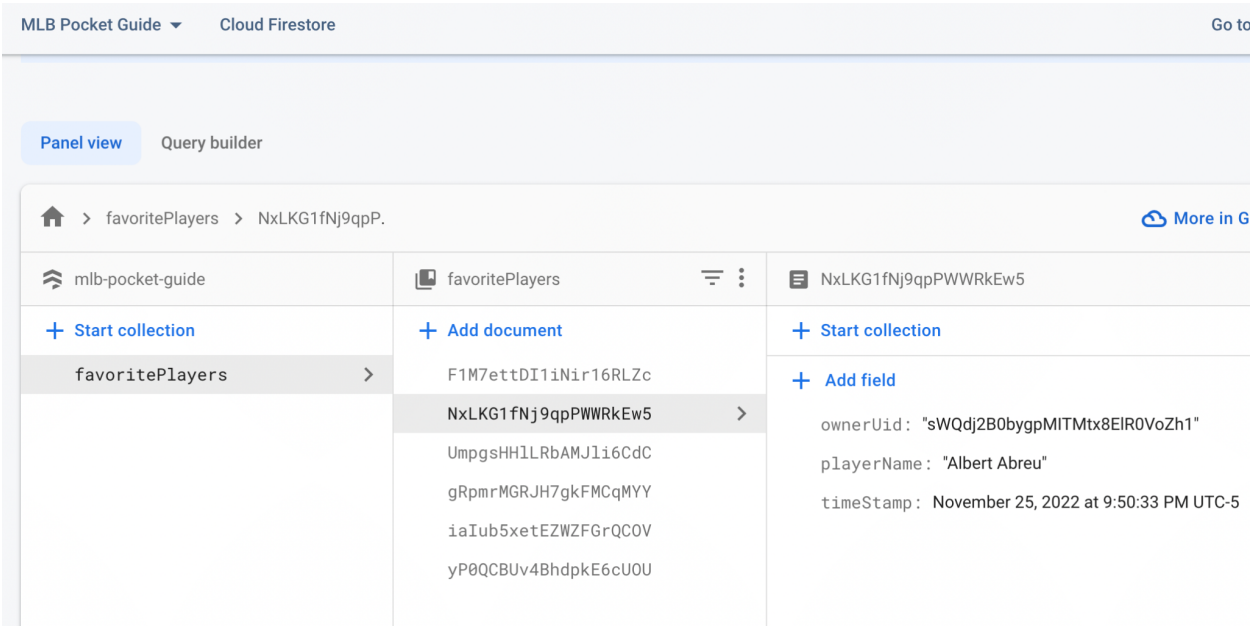
App Screenshots:




6:46
Search All Players
Choose a Player to Observe
Cory Abbott
CJ Abrams
Albert Abreu
Bryan Abreu
Jose Abreu
Domingo Acevedo
Ronald Acuna
Cristhian Adames
Willy Adames
Austin Adams
Chance Adams
Matt Adams
Players
6:45
Search Favorites
Choose a Favorite to Observe
Mookie Betts
Rafael Devers
Xander Bogaerts
Julio Rodriguez
Ronald Acuna
Clayton Kershaw
Bryce Harper
Players
6:49
Search All Hitters
Choose a Hitter to Compare
CJ Abrams
Jose Abreu
Ronald Acuna
Cristhian Adames
Willy Adames
Matt Adams
Riley Adams
Jo Adell
Ehire Adrianza
Jesus Aguilar
Nick Ahmed
Hanser Alberto
Matchups


6:46
Player Spotlight
SEARCH PLAYERS
FETCH DATA!
SELECT A FAVORITE
Rafael Devers
Age: 26 Position: 3B Hit: L Throw: R
Batting Outcomes
HRs
Hits
BBs
Ks
Outs
139
623
283
621
1,292
Performance Averages
AVG
0.42
0.35
0.28
0.21
0.14
0.07
0.00
2017
2018
2019
2020
2021
2022
Players
6:47
Player Spotlight
SEARCH PLAYERS
FETCH DATA!
SELECT A FAVORITE
Sandy Alcantara
Age: 27 Position: P Hit: R Throw: R
Batting Outcomes
69
524
263
638
1,424
Performance Averages
ERA
4.8
4.0
3.2
2.4
1.6
0.8
0.0
Players
6:47
Player Matchups
MOOKIE BETTS
SANDY ALCANTARA
FETCH!
Summary
0.293 Batting Average 0.223
13.7% Strike-Out Rate 21.9%
10.2% Walk Rate 7.7%
Hit Profile
36.6% Ground Ball Rate 50.0%
43.1% Pull Rate 41.8%
38.1% Hard Hit Rate 30.3%
Swing Profile
1.308 wFB/c 0.571
1.120 wOther/c 0.337
86.2% Contact Rate 76.4%
Matchups

Firestore Schema:



CLOC and Line Count:

The following table represents the command line output, though I should note that within the Kotlin section, 1 file worth ~35 lines of code is the boilerplate AuthInit.kt class that I borrowed from the assignments, and a second file has a repository built by hand (kind of like that one assignment we had) that is 1500 lines, 1 line per MLB player name. When you account for both of those, a much more reasonable number of Kotlin lines written is more like 1350 lines, which is in line with my inspection by hand.

Language	files	blank	comment	code
SVG	31	1	31	3401
Kotlin	15	186	124	2905
XML	66	112	38	2153
SUM:	112	299	193	8459

Code Frequency: