2020

Frisbee Golf Score Keeper

Sanil Apte
Grand Valley State University

Frisbee Golf Score Keeper

By
Sanil Apte

A project submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University
April 2020

Byron DeVries	4/21/2020
Project Advisor	**Date**

Table of Contents

Abstract

The project is a mobile application that will allow users to track scores while they play Frisbee Golf. The app is built for those like me who enjoy playing Frisbee/Disc Golf and would like to track their scores. In the mobile world there are two prevailing operating systems, iOS and Android. Both have their strengths and weakness, however, the problems that arise for developers is that both ecosystems require applications on their platform be in different coding languages. A solution to that is Flutter, which allows developers to build an app entirely in DART, Flutter automatically optimizes the applications so that they can work and be published in both ecosystems. Flutter itself is a new UI software development kit made by Google. While this app is currently been tested in the Android environments there is future pathway to iOS development already built in since it is built in Flutter. This app uses Google Firebase, a cloud-based solution which allows hosting and many other features for mobile apps and websites, for authentication services and data storage. This Frisbee Golf app supports multiple pages including the play page and scores page. The play and scores pages allow users to record and review their scores at their leisure.

Goals

Right off the bat there were a few features that I knew the application that I was developing must have. In my mind there were 4 core features, along with 3 side features.

Core features are:

- Login/logout
- Sign-up
- Track and saves scores
- Show the users history.

These core features are quite closely linked to each other as some of them would be useless without the other features already being implemented.

Side features are:

- Account info page
- Search for courses
- Search for discs page.

The Original Plan

Before the project semester began, I had done research into what tools I could use, and API's I could connect to, and much more to accomplish this project. At first, I reached the conclusion that I was going to use Apache Cordova using JavaScript and HTML to accomplish the project. I had also found a very convenient API built and provided by DG Course Review. Finally, I was going to put all the backend on Firebase which I had experience in prior. That all came crashing down just about a week into the project. The first blow was Apache Cordova. I wanted to build a cross platform app, and Apache Cordova advertised that it was able to do just that. However, I found out later that in order to build the iOS app you needed and Apple Mac device in order to

be able to even get off the ground in Cordova. I unfortunately do not have a Mac, so I had to look elsewhere. That was when I came upon Google's Flutter. Flutter also required that you have a Mac in order to deploy to iOS, but had the caveat that the iOS part could be added an anytime in the application's life cycle. It also added one more core goal to my project, the opportunity to learn a brand-new language, in an area that is quite new. Flutter uses Dart as its base programming language, and it was not a language that I had done any coding in whatsoever. So, this was an opportunity to pick up a new skill as well. The next blow was the DG course review API. I used Postman to try and see how the queries and results worked with this API. However, no matter how I tried, I received no data from the connection. Through some thorough research I found that the API was no longer available or supported and that any new connections were not allowed. Thus, the second blow. I had intended on using this API to allow users to search for discs and courses. There is a pretty well-known principle attributed to German General Helmut von Moltke that "No plan of operations extends with certainty beyond the first encounter with the enemy's main strength." While this plan was not war strategy, it followed that principle. Luckily, the last part Firebase did not let me down and that worked properly. Since my original plan had been changed so drastically, I created some detailed wireframes (shown in Figures 1 and 2), to get some concrete thoughts down on paper.

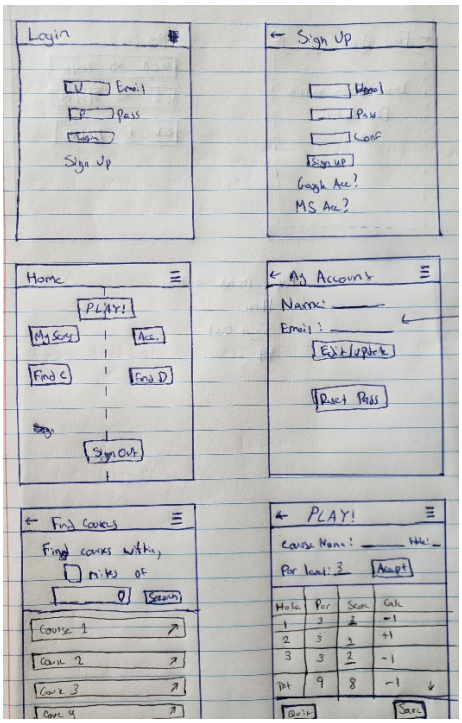

Figure 1: Wireframes 1

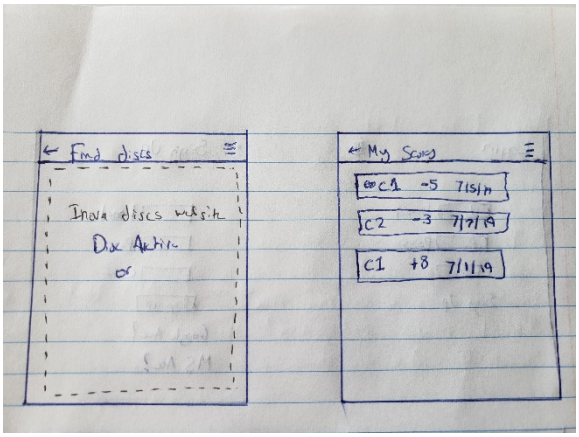

Figure 2: Wireframes 2

While a lot has changed from what the original wireframes were, the app functionality remains similar.

Development

This stage started off fairy smoothly. Since this was my first-time coding in Dart/Flutter, I dove into the tutorials that google had created along with some other tutorials made by other Internet users in order to familiarize myself with the language. I had hoped that Flutter would have a GUI editor that would allow a drag and drop sort of like how Android Studio does it for Java and Kotlin apps, but it unfortunately does not. However, even though there was no Visual Editor, the Flutter way of build GUI's was actually fairly simple and easy enough to work with. A lot of the tutorials that I listened to likened GUI development in Flutter to Lego Bricks. I absolutely agree, it is very much plugging different components on top of each other. An example of the code for widgets is in Figure 3.

```
Widget showPlayButton() {
  return new Padding(
      padding: EdgeInsets.fromLTRB(0.0, 45.0, 0.0, 0.0),
      child: SizedBox(
        height: 40.0,
        child: new RaisedButton(
          elevation: 5.0,
          shape: new RoundedRectangleBorder(
              borderRadius: new BorderRadius.circular(30.0)), // RoundedRectangleBorder
          color: Colors.green,
          child: new Text('Play!',
              style: new TextStyle(fontSize: 20.0, color: Colors.white)), // Text
          onPressed: (){
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => PlayPage()),
            );
          },
        ), // RaisedButton
      )); // SizedBox, Padding
}
```

Figure 3: showPlayButton

You can build as many of these widgets as you wish. Then following that you just have to add them to your build method. The way I accomplish it on different pages depends on the page. Some pages I directly have the widgets in the build because they are not too long or cumbersome, but other pages I have included a separate method that has all the widget calls held within it.

```
Widget showItems() {
  return new Container(
      padding: EdgeInsets.all(16.0),
      child: new Form(
        child: new ListView(
          shrinkWrap: true,
          children: <Widget>[
            showPlayButton(),
            showHistory(),
            showAcc(),
            showCourses(),
            showDiscs(),
          ], // <Widget>[]
        ), // ListView
      )); // Form, Container
}
```

Figure 4: showItems()

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Frisbee App"),
      actions: <Widget>[
        new FlatButton(
          child: new Text('Logout',
              style: new TextStyle(fontSize: 17.0, color: Colors.white)), // Text
          onPressed: signOut) // FlatButton
      ], // <Widget>[]
    ), // AppBar
    body: Stack(
      children: <Widget>[
        showItems(),
      ], // <Widget>[]

    ) // Stack
  ); // Scaffold
}
```

Figure 5: Build Call

Displayed in Figures 4 and 5 is an example of the second, where I call the showItems() method inside the build, which then goes and calls and stacks all the other widgets.

Overcoming Issues

As is normal, there a good amount of issues that I ran into during the development period of this app. Not including the issues I hit before development even began that is. The first one had to do with states. The app that I had created was one where it required users to login, thus there needed to be some sort of state included with the app. It was not until after I had created a login method that I realized that way I had created it, I could not access user information that was vital to saving and displaying information. I knew of the concept of states, but turning my app into a stateful one vs the current stateless turned out to be quite a bit harder than expected. Then I had to figure out what pages should be stateless, and which should be stateful. Currently I have all but two pages in the app as stateful and they do require the users ID in some fashion. The next issue I ran into was the firebase not properly storing the data I was sending to it. It was supposed to be saving three things, User Email, Course Name, and Final Score. That was not what was appearing the Firebase Realtime Database. Since the Realtime Database was not storing my information properly, I attempted switching the backend storage from the Realtime Database to the Firebase FireStore instead. It worked at first. However, when I tried to access the data later on to display it the app would crash. I switched back immediately, even if it was not saving properly in the Realtime Database, at least the app could read and receive data properly. I then proceeded to entirely rework, how the app was sending the information to Firebase. I created separate specific object called finalScores (pictured in Figure 6) which could format the data and turn into json format before sending the information.

```
class finalScores {
  String key;
  String course;
  int score;
  String userId;

  finalScores(this.course, this.userId, this.score);

  finalScores.fromJson(var value){
    this.userId = value['userId'];
    this.course = value['course'];
    this.score = value['score'];

  }

  String getCourse(){
    return course;
  }

  String getUserId(){
    return userId;
  }

  int getScore(){
    return score;
  }

  toJson() {
    return {
      "userId": userId,
      "course": course,
      "score": score,
    };


  }
}
```

Figure 6: finalScores Object

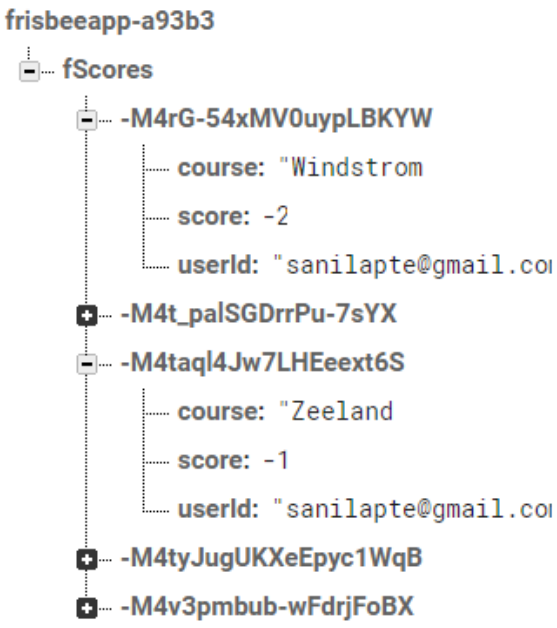

Figure 7: Firebase DB

This class was extremely useful later on, when I needed display the results on a different page. It allowed me to properly get the information from the Realtime Database (shown in Figure 7) and access and format it properly on the display screen.

Accomplished - Core

Of the original 4 core features that were outlined in my goals, I was able to accomplish all four of them. No matter the hitches that came up with them, I was able to get all of them working smoothly. Users can freely sign up, log in, and log out. The final two core features require user login and work without fault. Users can record scores, which are temporarily saved on the page

for the user's viewing pleasure (Figure 8). They can then save the information to the cloud Firebase Realtime Database and access that information at their leisure (Figure 9).

Figure 8: Scorecard

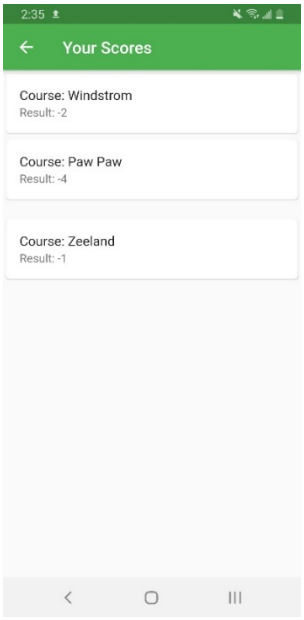

Figure 9: Scores

Accomplished - Side

There were 3 side features that I wanted accomplished, and while I did implement all three, not all of them are quite what I had imagined at the outset of the project. The Account feature was the only one that turned out how I had imagined it. It successfully allows users to manipulate different areas of their account such as their passwords. The final two features were the only that were majorly hit by the DG Course Review API being defunct. The features were the find courses and find discs. Both sets of information were going to be provided to the API. However, since that was not possible anymore, I tried a couple different things. At first, I tried including a live view of a different websites of disc golf manufacturers and retailer within my app itself, however that did not quite workout, due to how Android permission worked. However, I was able to links to the websites within the app that can launch the user's native browser. Then, for the find courses feature, I attempted to link my app with the Google Locations API and use that

search functionality to return results. I spent almost two weeks on that trying to fetch the data and show the top 5 results. However, it would not do as I asked. So, I ended up doing what I had done for the find discs page and included links to different sites which have disc golf course locations and reviews. This last feature is one that I am going to continue working on even past the end of the semester and get it working. Many times, I thought I was close to having it working but it did not. Thus, the last two features were "completed." (Figures 10 and 11 show the features in their current state)

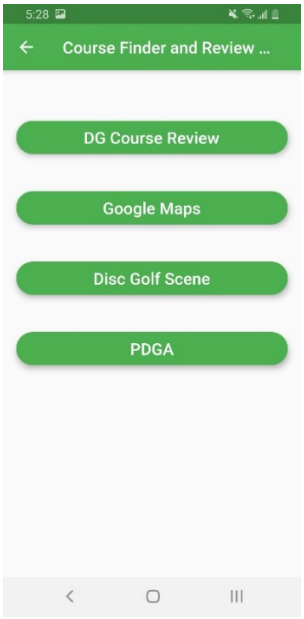

Figure 10: Course Finder

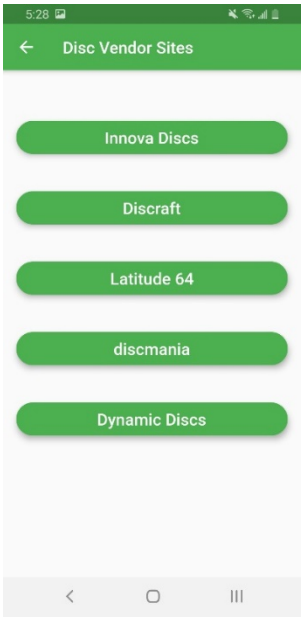

Figure 11: Disc Vendor

Flutter/Dart

One of the goals that I did not mention in the accomplished section was the last one that was added to my goals list. The goal of learning Flutter/Dart. The tool and language were not like anything that I had really worked with before in a mobile environment. In the mobile development class that I had taken, I learned how to build an Android and iOS app using Java and Swift. In both those languages I found that apps were given functionality after the UI had

been designed. For Flutter, I found that quite a few times you are building functionality alongside of building the UI. I believe this is due to the functionality being built into the widgets themselves. Now that I have worked in Flutter/Dart for the past three months, I see opportunities for new features and other things that I can implement. Also, since I have grown there are things, such as the FutureBuilder widget, I now know about Flutter that will allow me to go back and optimize some of the code that I created at the very beginning of the project. The FutureBuilder is a great way to display async operations to the users. At the very beginning of development, I had been calling async methods from standard widget, then calling the setState() to force a refresh of the screen. The FutureBuilder gets rid of the need for that method entirely as the entire widget itself is an asynchronous method. Thus, there is no need to call other methods or even setState() as that widget does everything before load even occurs.

Flutter has a bunch of cool features like that, another one that I quite enjoyed making use of was the Hot Reload feature. The Hot Reload allows a developer to make changes to app code while the app is still running, then hitting the Hot Reload button has those changes pushed and mimicked in the live app. No interruption, this is useful especially if a developer is running off of an emulated android phone because it can be quite slow to stop, reload, and restart an app completely. However, there are some downsides to using flutter and dart. I have already mentioned the lack of GUI designer, but that is not the only hang up I found. For one, running an app in debug mode can be a bit tedious. The debug mode does not work at all with the Hot Reload and at time will cause the emulator to crash. I did not dare test it with my own phone after this occurred on the emulator. At other times it just felt a little unwieldy compared to the some off the other debuggers that I have used. The second knock against that I found against Flutter is the lack of Open Source Libraries. Compared to one of its competitors React, the total

amount of external libraries is paltry. React is a JavaScript framework, as such there are a ton of Node.js modules and packages that a React developer can use.

Coming back to the original question, have I accomplished the goal of learning flutter. I would say that I have, but that I am not done. This past semester I have immersed myself in learning this new technology and I have found it fascinating. There is always give and take when it comes to technology, and I found that what Flutter "gives" is worth more than what it "takes" away. There are also things I have yet to discover about Flutter and I am looking forward to figuring them out.

Conclusion

This project was one I immensely enjoyed working on. I spent quite a lot of time on it this past semester. Between learning a new language, troubleshooting issues, and finishing features there was not a dull moment. At the end of the Master's Project, I am quite happy with what I accomplished. It is not perfect by any definition of the word. There are still a couple bugs that I have yet to iron out and figure from where they are occurring. I suspect they are caused by some hidden race conditions because they do not happen consistently. Of the 7 or so feature goals that I set, I have fully accomplished 5 of them and partially accomplished the last 2. Learning Flutter has also been a great boon to my Resume, as I have had a couple job interviews offered to me because I had knowledge of what Flutter is and a little bit of experience in it. This mobile application is one that I am going to continue working even after graduation and starting a job. I am already planning some of my next steps, such fleshing out the Find Discs and Courses page. Also, I have ideas for what future features could be, such as member levels and party play.

Bibliography

"Documentation | Firebase." *Google*, Google, firebase.google.com/docs.

"Flutter Documentation." *Flutter*, Google, flutter.dev/docs.

"Learning Center" *Postman,* Postman, https://learning.postman.com/docs/postman/launching-postman/introduction/

Moltke, H. G. von. (1993). *Moltke on the art of war: Selected writings*. Translated by Daniel J Hughes and Harry Bell. New York: Presidion Press.