

Malila Freeman  
Professor Davison St. Germain  
CS6019  
December 9, 2022

## **Introduction**

For many golfers, keeping track of their overall game score and score per hole simply isn't enough to give valuable insight into their gameplay. Strokes gained offers a more complex way to analyze a golfer's performance by providing a per-stroke calculation that compares their stroke to the PGA Tour benchmark. This calculation tells the player whether they lost shots (a negative number) or gained shots (a positive number) compared to the benchmark.

Currently, there are very few platforms that offer strokes gained calculation capabilities, and only one that offers real-time strokes gained calculations. This platform, called Strokes Gained Golf, was patented by my dad and uncle a few years ago. However, as of now, the real-time calculations are only offered via a website, meaning that users must log into their web accounts using a browser on their mobile device in order to access the real-time calculations that makes strokes gained golf so enticing. My app offers a solution to this problem by providing a way for users to access Strokes Gained Golf functionality with an app on either an Android or an iOS device.

## **Background**

The current web-based platform for Strokes Gained Golf is built well, but it introduces unnecessary hurdles for users because it is not as intuitive to use its features in a mobile web browser as it is to use them in an app. Importantly, other apps centered around strokes gained do not have patents that allow them to offer real-time strokes gained calculations to golfers while they're on the course, so they do not offer users this appealing feature at all. Since this real-time capability is the basis of the patent for Strokes Gained Golf (and therefore the main appeal of the platform), it makes sense to offer an app in addition to the web-based platform. Because of its unique patent, the Strokes Gained Golf platform inherently has a big edge in the market, but I believe that the lack of an app is slowing its progress and making it ultimately less appealing compared to other strokes gained platforms. Ultimately, then, my main motivation for developing the Strokes Gained Golf app for my capstone was to attract more users and make them more likely to subscribe to the service because they wouldn't have to take as many steps to access its most distinguishing feature.

Developing the Strokes Gained Golf app has offered a unique opportunity not only to design a different version of a platform that already exists, but also to develop software according to someone else's specifications. I needed to take the existing features from the website and transfer them into a format that made sense for a mobile app while listening and

responding to input and suggestions from my uncle. I was particularly drawn to this project because I wanted practice with both tasks. I felt that developing software for a pre-existing platform would provide good practice for when I interview for a position and join a company later on, and I knew that this project would be a great opportunity.

As a starting point for designing the Strokes Gained Golf app, I knew that I wanted the interface to be simple and direct while also offering a few extra helpful features. Therefore, in addition to the ‘Start New Game’ feature which allows the user to input a course name, time, date, number of holes played, and an option to favorite the course and then input the necessary information for each stroke played, I also implemented a ‘Favorite Courses’ feature which allows the user to see all of the courses they have favorited on one screen, and I implemented a ‘View Past Games’ feature which allows the user to review their statistics from previous games. Overall, since the purpose of the Strokes Gained Golf app is to be used on the golf course with every stroke the user takes, I wanted to make sure that the app flows with their golf game and that it is always clear what inputs are expected.

### **Solution Description**

The Strokes Gained Golf app is designed for iOS and Android devices and displays users’ strokes gained calculations in real time. It also stores these calculations so that they can review them later. Currently, the web version of Strokes Gained Golf has about 300 users, and I also wanted to plan for some growth. With these specifications in mind, I decided to develop the app using Flutter, which builds apps that run on multiple devices from a single codebase. I also used Google Firebase to authenticate users and Cloud Firestore to store their strokes gained data.

### **Class Structure**

In Flutter, the entire app that is built is represented by a widget tree ([Citation](#)). The Strokes Gained Golf app is launched from `main()`, which runs stateless widget `MyApp()`. This is a `MaterialApp` widget, which establishes design elements of the app as well as the app’s Home: for Strokes Gained Golf, the home page is wrapped in the `MainPage` widget which loads either the sign in page or the home page of the app, depending on whether the user is signed in or not. A full class diagram is provided in image 1 (below).

## Class diagram

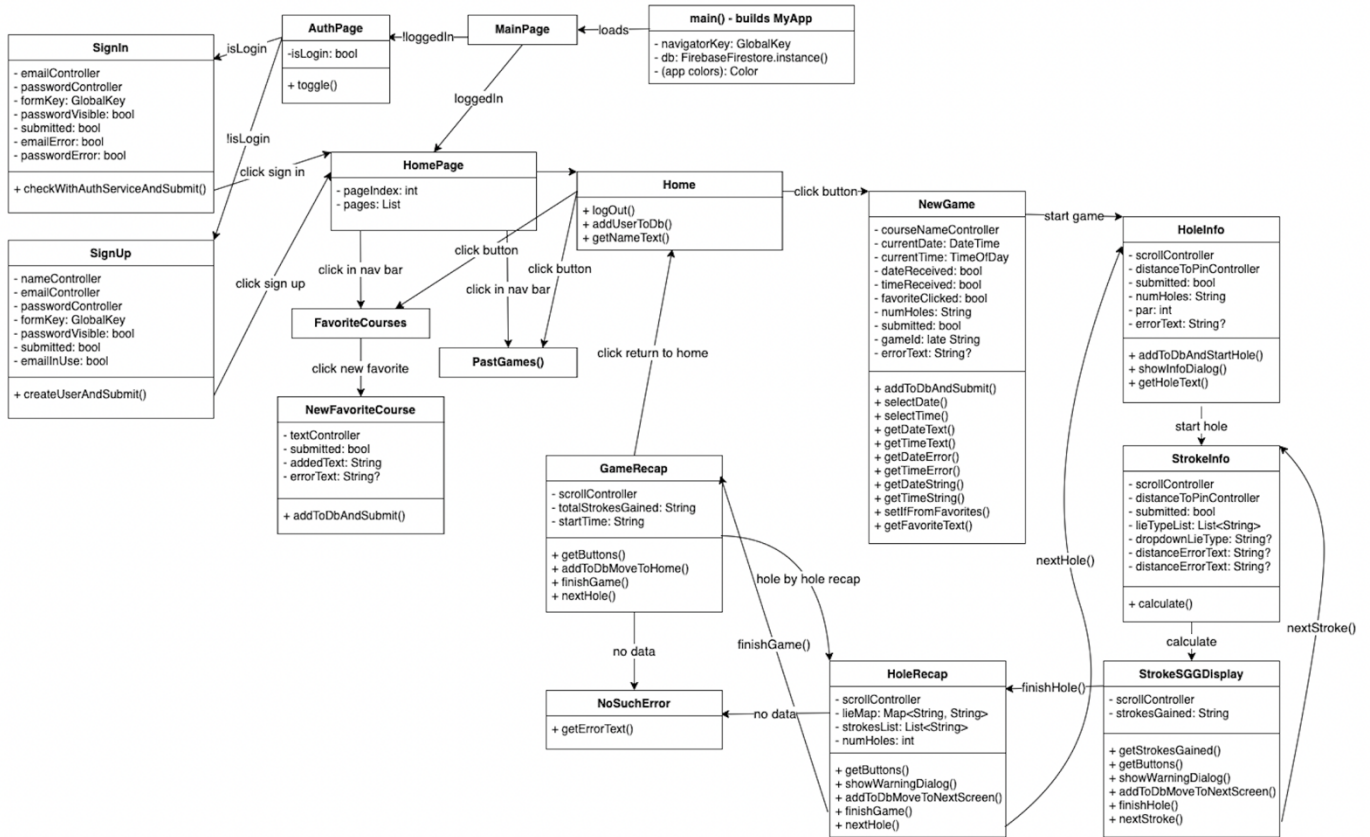
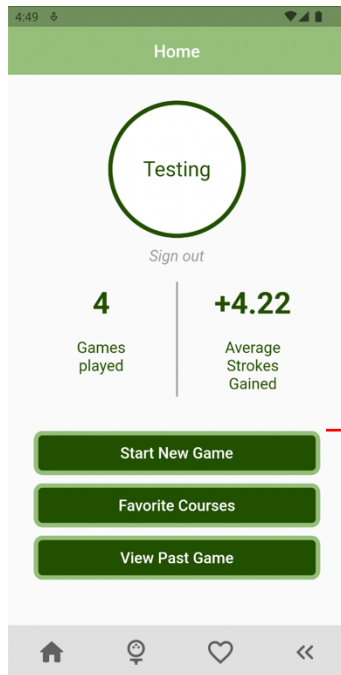


Image 1.

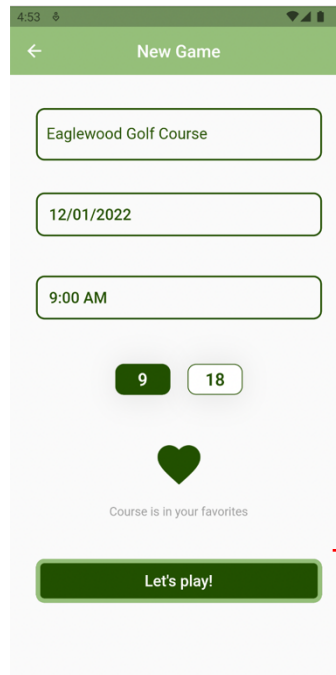
## App Flow

The flow of the rest of the app, as illustrated in the diagram, is as follows: Once the user enters the home page, they can select from a few options: Start New Game, Favorite Courses, and View Past Game (image 2, below).

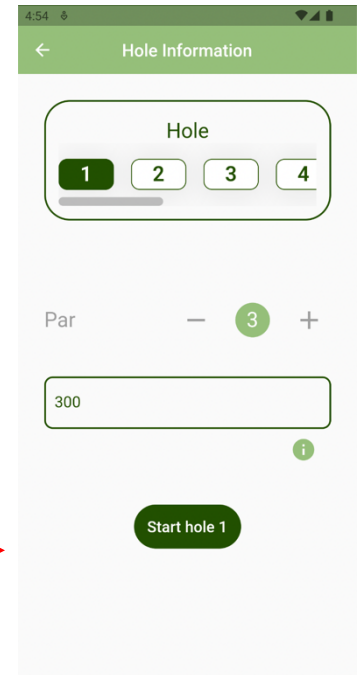
By clicking on Start New Game, they are taken to a New Game page in which they can input the course name, date and time, specify whether they are playing 9 or 18 holes, and favorite the course (image 3, below). Once they click 'Let's Play,' they are taken to the Hole Information page in which they indicate the par of the hole and the initial distance to the pin from the tee (image 4, below).



*Image 2.*



*Image 3.*



*Image 4.*

This information will be used to calculate their strokes gained statistics. Once they click ‘Start Hole 1,’ they are taken to the Stroke Information page in which they again indicate their lie type and distance to the pin (image 5, below). This information is inputted after the first stroke, as both a start and end lie type and distance are required for the calculation. Once they click ‘Calculate,’ they are presented with their strokes gained calculation for that stroke (image 6, below). They then have the option to either proceed to the next stroke or end the hole. Ending the hole early is acceptable, but will result in incomplete statistics; therefore, an alert pops up if this is attempted. If they click ‘Next Stroke,’ they will continue with entering their lie type and distance until they select ‘in the cup’ as their lie type and finish the hole. Once they click ‘Finish Hole,’ they will be taken to the Hole Recap page where they can see all of their strokes as well as their score and strokes gained for the hole (image 7, below). At this point they have their option to either proceed to the next hole or end the game (again, ending the game early is acceptable, but will result in incomplete statistics, thus prompting an alert message). If they click ‘Next Hole’ they will continue entering their lie type and distance information as with the previous holes (image 4, above).

Stroke Information

Hole

1 2 3 4

Par 3 Stroke 1

Fairway

100

Calculate

Image 5.

Stroke Strokes Gained

Hole

1 2 3 4

Par 3 Stroke 1

-0.09  
strokes gained

Finish hole Next stroke

Image 6.

Hole Recap

Hole

1 2 3 4

Stroke	Start Lie	End Lie	Strokes Gained
1	Tee	Fwy	0.09
2	Fwy	Grn	0.37
3	Grn	Grn	-0.56
4	Grn	Cup	0.00

even on par -0.10 strokes gained

Image 7.

Once they click 'Finish Game,' they will be taken to the Game Recap page where they can see their course, date and time of play, game duration, score, and strokes gained, as well as view their game by hole (image 8, below). From here they can return to the home page.

Game Recap

Eaglewood Golf Course

12/01/2022 at 9:00 AM

8 hours and 6 minutes

2 over par +0.12 strokes gained

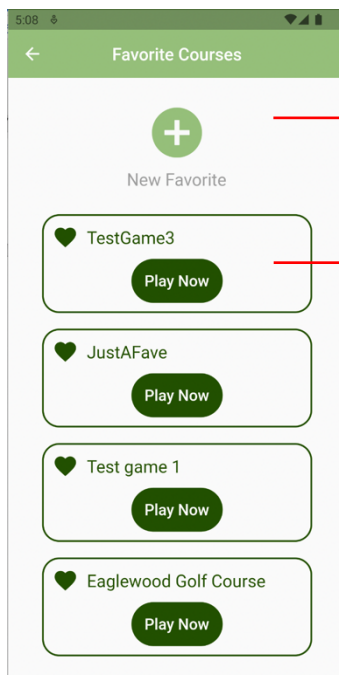
Hole

6 7 8 9

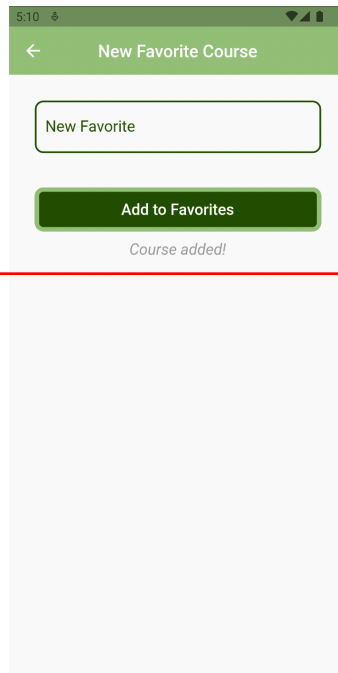
Return to home

Image 8.

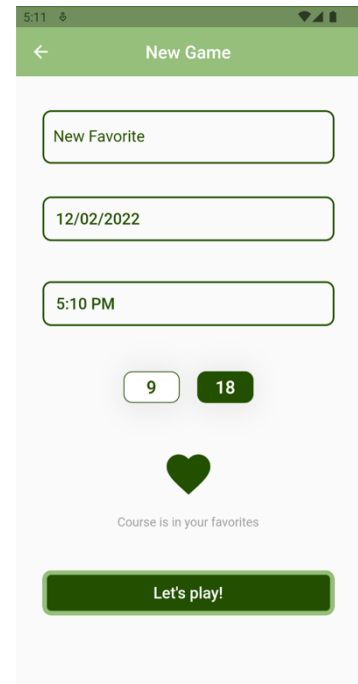
By clicking on Favorite courses, the user is taken to a page that lists all their favorite courses (image 9, below). They can add a new favorite course which will take them to a page where they can add a new favorite (image 10, below). They can also select 'Play Now' on one of their already existing favorites which will take them to the New Game page and autofill the course name, date, and time (image 11, below).



*Image 9.*

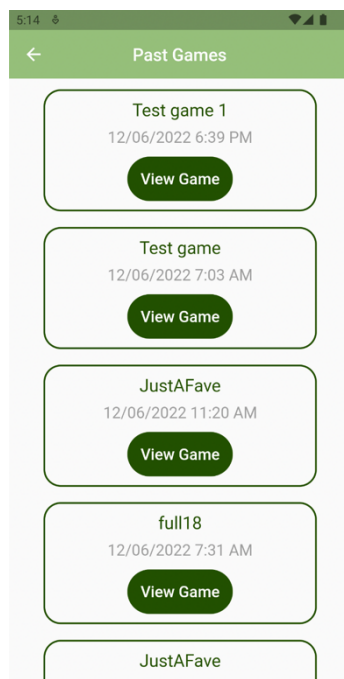


*Image 10.*



*Image 11.*

By clicking on View Past Games, the user is taken to a page that lists all the games they have played with the course name, date, and time (image 12, below). From here they can click 'View Game' which will take them to the Game Recap page for that game (image 8, above)



*Image 12.*

## Front End and UI

When I began developing my app, I decided to start with the front end. I wanted to get a good idea of how the different pieces worked together, and I also wanted to get approval from my uncle on the UI before adding in back-end functionality.

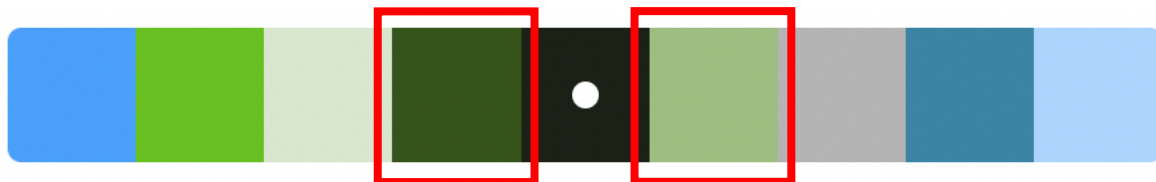
### *Overall Design*

The first step I took towards this goal was to outline what I wanted my app to look like. I knew that I was fairly limited on design since I do not have access to a graphic designer, so I designed my app's basic look using Microsoft Word and I kept all the widgets simple. For the widgets, I used colors of green that were present in the Strokes Gained Golf logo (image 13, below).



*Image 13.*

Specifically, I incorporated the colors outlined in red in image 14 (below).



*Image 14.*

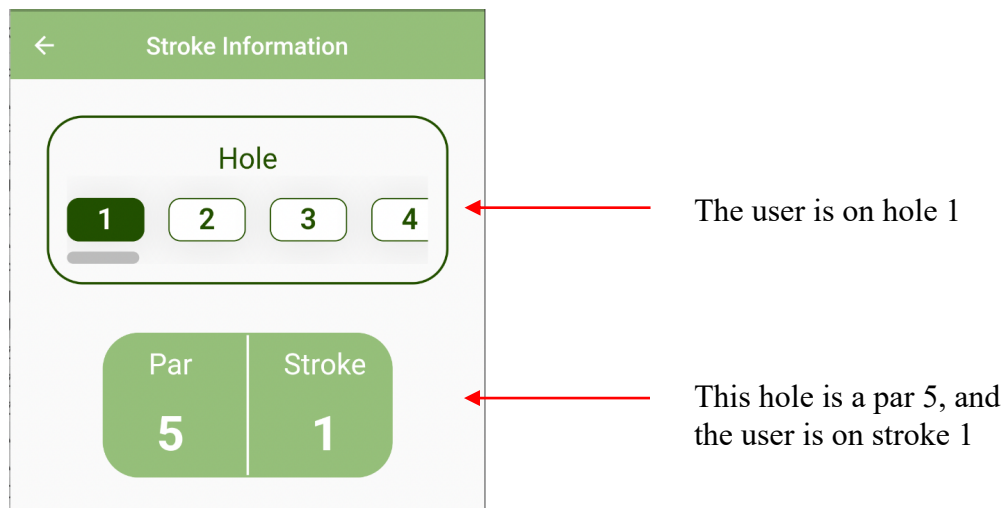
I decided to use the shades of green instead of other colors because I felt that the greens did the best job of emulating a golf course, and I wanted the overall feel of the app to be consistent with its purpose.

### *User Flow*

When designing the flow of the app, I knew that I wanted the UI to be consistent with the golfer's experience while playing a game. Therefore, I created separate routes (screens) for every step of the strokes gained calculation process. I also wanted to ensure that it was always clear to



the user which hole and stroke they were on, as well as the par of the hole, so I made sure to include widgets that specified this on each route (see example in image 15, below).



*Image 15.*

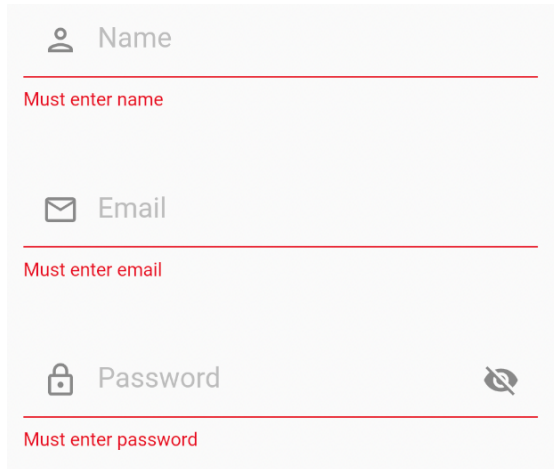
Another critical component to the user flow was establishing accurate error messaging. To achieve this, each widget that required error messaging was built using a `TextFormField`. This widget takes a validator in which you can return specific error messages for specific cases. For example, the sign up page validator looks like this:

```
validator: (text) {  
  if (text == null || text.isEmpty) {  
    return 'Must enter email';  
  }  
  if (!EmailValidator.validate(text)){  
    return 'Invalid email';  
  }  
  if (_emailInUse){  
    return 'An account already exists for that email';  
  }  
  return null;  
},
```

No error messaging will appear until the user clicks on the page's version of a 'submit' button. Then, if anything is wrong with their inputs, the error messages will appear and adjust as the user re-types their entry. Additionally, if there are errors with the user's entry(s), they will not be able to move to the next page.

Error messaging was critical in several places. On the sign-up page, users must enter all of the fields (see image 16, below). They must also enter a valid email (see image 17, below) and password (see image 18, below). If a user has already created an account with the email entered, they will also be notified (see image 19, below).





Name

Must enter name

Email

Must enter email

Password

Must enter password

Image 16.

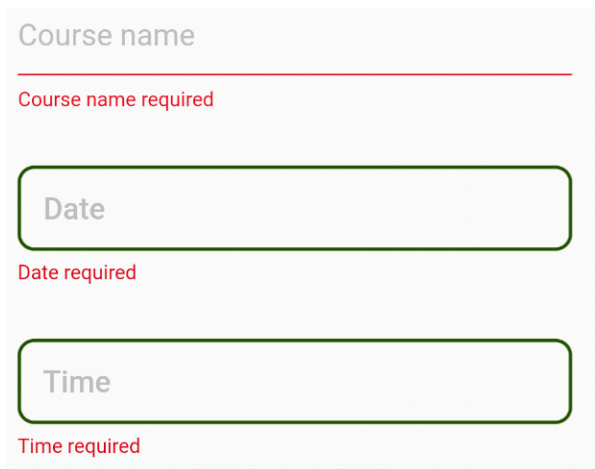


example@

Invalid email

Image 17.

Similar error messages occur on the log in page. Users must enter all the fields and must enter both an email and password that match an already existing account. For the New Game page, users must enter a course name, date, and time to proceed (see image 20, below). This ensures that if the user wants to look back on the game later, all of that information will be available. It also allows for the course name, date and time played, and game duration to be displayed with the game summary once the user has completed their game (see image 21, below) without any errors.



Course name

Course name required

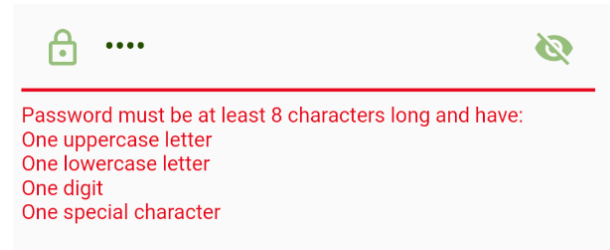
Date

Date required

Time

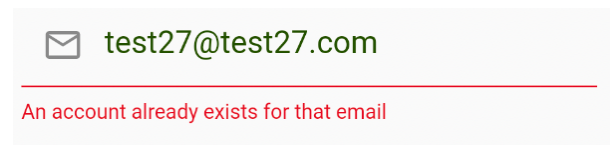
Time required

Image 20.



Password must be at least 8 characters long and have:  
One uppercase letter  
One lowercase letter  
One digit  
One special character

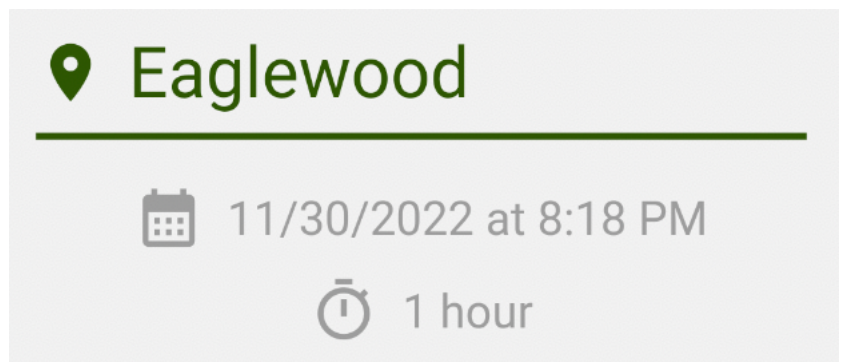
Image 18.



test27@test27.com

An account already exists for that email

Image 19.



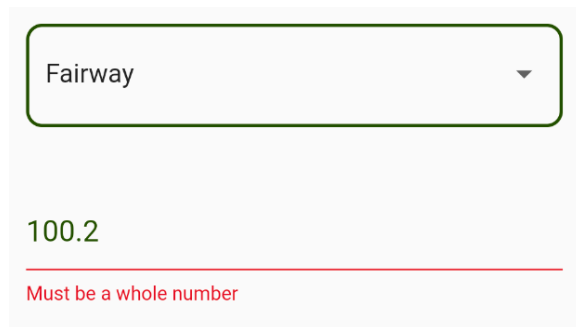
Eaglewood

11/30/2022 at 8:18 PM

1 hour

Image 21.

Error messaging was particularly important for the strokes gained calculations themselves, in which the user must enter their lie type and distance to the hole. The pre-determined tables that are used for these calculations are very specific, and so certain inputs will yield an invalid calculation. The user, at baseline, must enter both a lie type and a distance. Since I decided to let the user input their own text for distance to the pin (a spinner wasn't practical since they can enter anything from 0-600 yards), they must enter a number with no spaces or letters. Additionally, distances that the user can enter are specific to the lie type they have entered. For example, if they are not on the green and try to enter a decimal number, they will get an error (see image 22, below).

The image shows a user interface for a golf application. At the top, there is a dropdown menu with the text 'Fairway' and a small downward-pointing arrow on the right. Below this, there is a text input field containing the number '100.2'. A red horizontal line is positioned below the input field, and directly beneath it is the error message 'Must be a whole number' in red text.

*Image 22.*

A few additional features also help with user flow. For example, if the user selects 'in the cup' as their lie type, it means they have made it in the hole and their hole is complete. Thus, the distance field automatically fills to 0 (see image 23, below). If the user has not completed the hole, once they view their strokes gained statistic, they have the option to move to the next stroke or finish the hole early (see image 24, below). However, once they have selected the 'in the cup' lie type, it means that their hole is complete, and they should not be moving on to the next stroke. Therefore, this button is then not offered (see image 25, below).

In general, these choices were made to improve the user experience with the app and help ensure that everything goes smoothly during their gameplay.

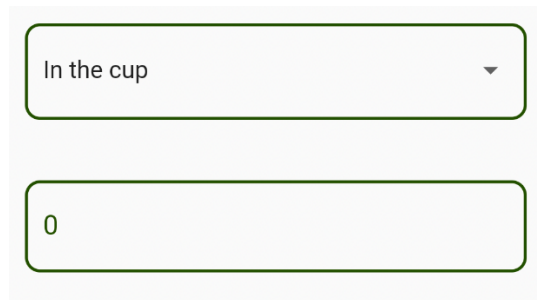
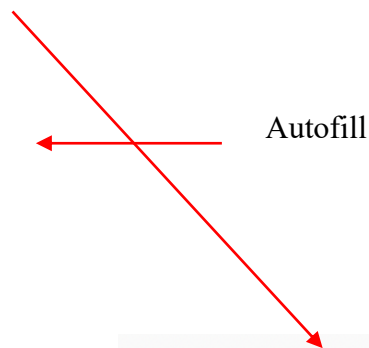


Image 23.



If the user selects 'in the cup', the next screen will not have the option for 'next stroke'

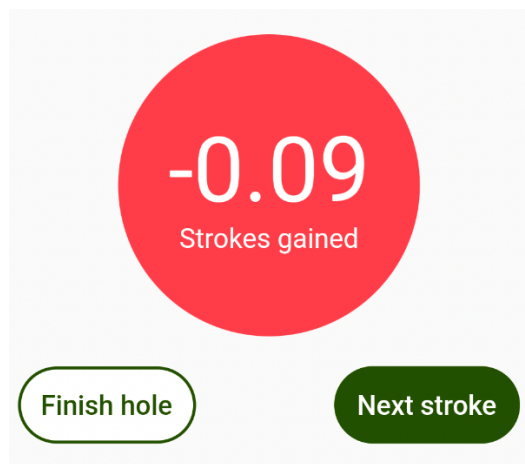


Image 24.

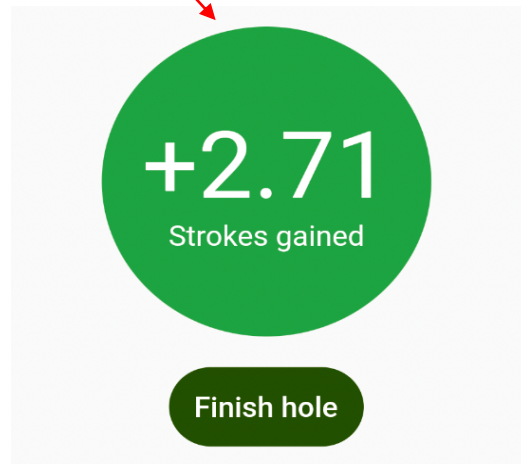


Image 25.

## Code Structure and Widgets

In Flutter, each route (or screen) is simply a widget. Therefore, I designed each independent screen in my app to be a stateful widget. Stateful widgets allow for the screen to change with user interaction, and they are also compatible with database use. Since all of my screens require either user interaction or fetch data from Cloud Firestore (or both), I needed to use stateful widgets.

As mentioned previously, a Flutter app is represented by a widget tree. This means that for any route that is built, what is displayed to the user in the UI is made up of several nested widgets. While my main routes are stateful, many of the widgets that I display to the user are stateless (meaning they don't change in response to user or database interaction). Therefore, I abstracted many of the stateless widgets into a "WidgetHelpers" folder in my project. This was helpful for many reasons: it allowed for reuse of the widgets in multiple routes, it allowed for the nested structure of the main route widgets to be less complex, and it ensured that whenever I changed a design component of one of the stateless widgets, that this change was reflected across all the routes that utilized it. For example, in image 15 (above), the widgets that display the hole, par, and stroke are all stateless widgets that were abstracted to the WidgetHelpers folder.

## Authentication

Since Strokes Gained Golf is designed for individual users, it was important to develop a sign-up page as well as a log in page, and display these pages appropriately according to whether the user is already logged in. To achieve this, I used Firebase Authentication. Firebase Authentication offers methods such as *createUserWithEmailAndPassword()*, which I used for my sign-up page, and *signInWithEmailAndPassword()*, which I used for my log in page. In order to accurately switch between the log in, sign up, and home pages, I created a MainPage class, which handles which page to load according to the user's authentication status. This MainPage class is established as the "home" of the MyApp widget because whichever page it loads is the one that will first appear to the user. Flutter offers a StreamBuilder widget, which builds itself based on a snapshot of an interaction with a stream ([Citation](#)). In this case, that stream is the Firebase method *authStateChanges()*, which is a listener that fires when the user's authentication state changes. Therefore, the pseudocode for the MainPage widget looks like this:

```
body: StreamBuilder<User?>(
  stream: FirebaseAuth.instance.authStateChanges(),
  builder: (context, snapshot) {
    if (/*snapshot is waiting on something*/) {
      // return a progress indicator (loading symbol)
    }
    else if (/*snapshot has an error*/) {
      // return error widget
    }
    else if (/*user exists and is logged in*/) {
      // return home page
    }
    else {
      // return AuthPage
    }
  }
),
);
```

With this, the MainPage widget is able to successfully handle which page to load when the user opens the app by listening to the user's authentication state through the *authStateChanges()* stream.

The two authorization pages, the sign up page and the log in page, call the *createUserWithEmailAndPassword()* method and the *signInWithEmailAndPassword()* method, respectively. When the user presses the "Sign in" and "Sign up" buttons, an asynchronous function containing one of these methods is called. Importantly, if method fails and therefore the user is not signed up or signed in, the exception is caught, and an error message is displayed for the user. For example, if the user tries to log in with the wrong email or password, the specific 'wrong-password' exception is caught and sets a Boolean in the widget, which then displays the error under the password field. Similar logic exists for the sign-up page. The pseudocode for logging in a user thus looks like this, and is called from the "Sign in" button:

```

try {
  // Sign in with firebase authentication
};
}
on FirebaseAuthException catch (e) {
  if (e.code == 'user-not-found') {
    setState(() {
      _emailError = true;
    });
  } else if (e.code == 'wrong-password') {
    setState(() {
      _passwordError = true;
    });
  }
}
}

```

Importantly, the thrown exception means that the authorization state of the user does not change, and thus the home page is not loaded and the user must attempt to sign in or sign up again.

## Back End and Database

Once my front end was in place, I had a good idea of what pieces of information I needed to store for each user and how they related to each other. Since I planned to integrate a backend, I was very intentional about pieces of data I needed, data types, and parameterizing helper functions and widgets. This helped me to initially design my database, and implementing it was straightforward since my widgets and routes had previously been designed to handle these specific types of information.

### Design

When writing out an ER diagram for my database, the nested structure of my data became highly apparent. Users play games, which are made up of a number of holes, which are each comprised of a certain number of strokes. This structure is outlined in the ER diagram in image 26 (below).

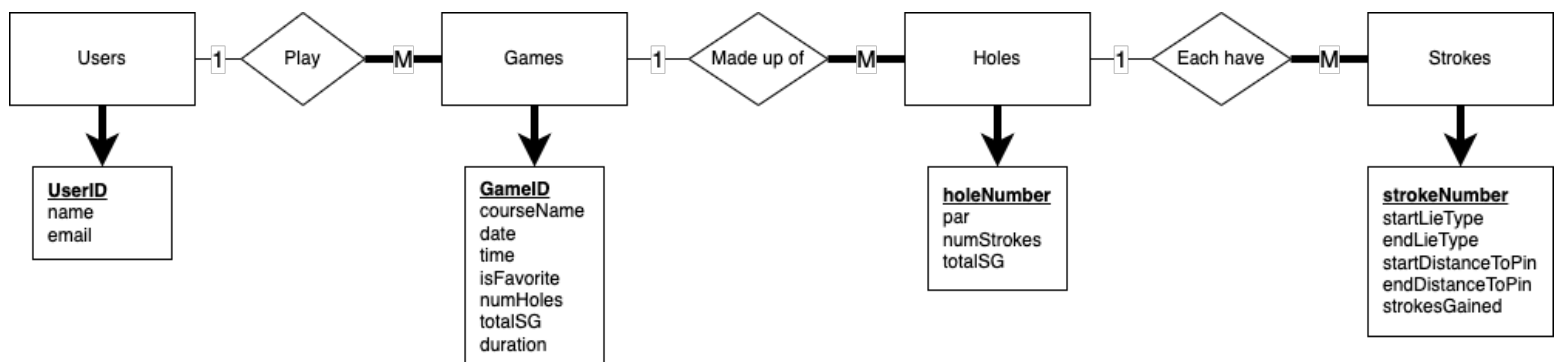
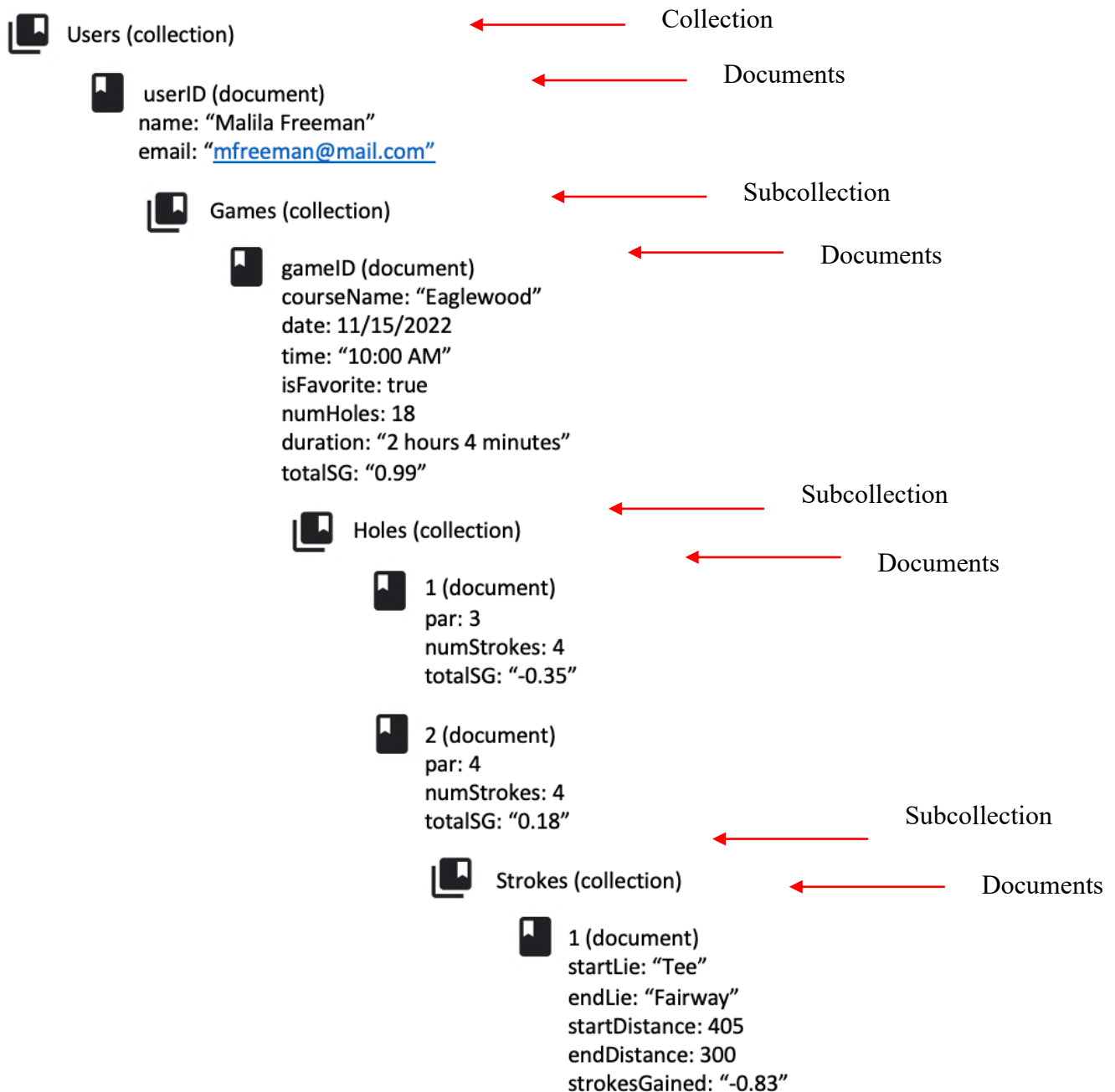


Image 26.

Cloud Firestore databases can be organized in hierarchical structures, such that that inside of collections, data is stored in documents and within those documents there can be subcollections. The example that is provided in the documentation for a database built with subcollections aligned well with the database I wanted to build, so I created a diagram for my database using collections, documents, and subcollections (see image 27, below). Therefore, before I even started implementing my database, I had a solid concept of what I would need to store and its structure.



## Code Structure

**Create.** To get the necessary information into the database, there are several points throughout the app where I must add a new collection or document. For the most part, these creates are done with the click of a button; for example, when the user clicks “Let’s Play” from the New Game screen, a new game collection and its corresponding document is added with courseName, date, time, isFavorite, and numHoles according to the user’s entries and before routing to the next screen. Similarly, a new hole collection and document is added when the user clicks “Start Hole” on the HoleInfo screen and a new stroke collection and document is added when the user clicks “Next Stroke” on the StrokeInfo screen. The pseudocode looks like this (and is called on the onClick() method of the button that is being clicked):

```
void _addToDbAndSubmit() {  
    // get necessary data from the UI  
    // get the current user that is logged in and their userID  
    // get a reference to the collection using the userID and other identifiers  
    // create a map with the data to be entered  
    // add map to the database, then navigate to the next screen  
}
```

**Read.** A database is needed for the Strokes Gained Golf app so that they can see summary/recap information when they complete a hole, complete a game, or view a past game, and so that they can view their favorite courses and play a game from there. Therefore, the information that is read from the database is displayed in widgets, which requires the use of a StreamBuilder. In flutter, the StreamBuilder widget builds other widgets based on snapshots that it receives from a stream ([citation](#)): in this case, a query from the database. In order to query the database, a reference to the collection or document with a snapshots() call creates either a QuerySnapshot or a DocumentSnapshot, respectively. These are given to the StreamBuilder, which can then access data from the query and build the widgets accordingly. The pseudocode looks like this:

```
StreamBuilder<QuerySnapshot>(  
    stream: // query,  
    builder: (context, snapshot) {  
        if (snapshot.hasError) {  
            // return error widget  
        }  
        if (snapshot.hasData) {  
            // access data from query  
            if (/* any data types are null */) {  
                // return a widget displaying that the information is not  
                available  
            }  
            // return widget displaying the data  
        }  
        // return a progress indicator  
    });
```



Notably, if the data does not exist or something goes wrong with the query, widgets are built that display the error to the user.

**Update.** There are document fields that need to be updated once more data is obtained, such as adding the game duration and total strokes gained to a game's document once the game has been completed and adding the number of strokes and the total strokes gained for a hole once the hole has been completed. These updates are also done with button clicks, and they are very similar to creates except that a reference to the document (rather than the collection) must be obtained. Then, instead of just calling "set" on the database, the set function with "merge: true" is called, which adds the document field without overwriting the entire document.

## Strokes Gained Calculations

Arguably the most important functionality of the Strokes Gained Golf app is to calculate the user's strokes gained statistics correctly. The strokes gained equation itself is quite simple: for each stroke, the calculation is: **(Starting Position Strokes Gained – Finishing Position Strokes Gained) – 1**. However, the more difficult aspect is getting the starting and finishing strokes gained positions. These are obtained from a table, which is based on the baseline number of shots that PGA golfers take to complete the stroke. I received this table from my uncle in PDF format, and to calculate strokes gained in my own app, I converted the PDF tables into maps of distance mapped to baseline number of shots for each lie type. Then, I calculated the starting and finishing position strokes gained by looking up the distance in the table for either the starting or ending lie type. However, not all distances are included in the table; the distances are in varying increments, depending on the lie type. Therefore, I wanted to make sure that I got the most accurate value as possible by performing linear interpolation, which gets an interpolated value for any distance (even if it is not listed explicitly in the table). To do this, I had to find the lower and higher multiple of the desired distance (for example, if the table is in distance increments of 10 and I want the distance for 15, the lower multiple would be 10 and the higher multiple would be 20). I then had to find the interpolation factor, which gives where in the distance increment the desired distance lies (for example, if the distances in the table are 10 and 20 and I want to get the value for 15, the interpolation factor would be 0.5). I then had to get the baseline shots from the table for the lower and higher multiple, then perform the linear interpolation equation: **lowerMultiple \* (1 – interpolationFactor) + secondMultiple \* interpolationFactor**). Once I understood what I was doing, this was straightforward. The difficult part was adjusting for the different increments in the different lie type tables and writing a generalized interpolation function that worked for all of them.

The resulting strokes gained calculation is either positive, meaning that the player gained shots compared to the baseline, or negative, meaning that the player lost shots compared to baseline. To display this intuitively to the user, I colored the widget green if their strokes gained calculation was positive, and red if their strokes gained calculation was negative.

## Timeline and Project Issues

Overall, my actual week to week work lined up well with my proposed timeline, although I did do a few things differently. Primarily, in my initial timeline I had planned to work on the front end and back end simultaneously, when realistically I ended up almost entirely completing

my front end before implementing the back end. I'm happy with this choice, though, because by completing the front end first I had a better understanding of what data was needed where, which made designing and implementing my database much easier.

Additionally, I had a few blockers that made completing other back-end components, such as the strokes gained calculations, more difficult. Communicating with my uncle and organizing meetings took longer than I thought, and so there were some materials that I needed but didn't immediately have (such as the strokes gained tables), which prevented me from making fast progress. Additionally, in my initial timeline I had planned to at least attempt to get the golf course API working (so that the user would not have to input their own lie type and distance with every stroke, and it could instead be automatic), but other pieces of development ended up taking longer than expected and so I never got to this feature. However, I had a feeling from the beginning that this might be the case, so I did have the API integration listed in my proposal as a stretch goal. Additionally, I had planned on having a 'Statistics' section that displayed the user's overall strokes gained statistics with different visuals (such as graphs, charts, averages over time, etc.), but I also did not end up getting to this feature. In my initial proposal this was listed as "high priority" but not "top priority", but I did get to my other "high priority" feature (viewing past games).

I think the main reason for this project taking longer to complete than expected was my initial learning curve with Flutter. I really struggled to grasp the nested structure and the widget creation in the beginning, and so designing my front end took a lot longer than I had initially expected. I am proud of how far I've come, though, because now Flutter comes naturally to me and as I've added front-end features later in the process, it has gone much faster than when I started off. Additionally, handling errors with database fetching was more complex than I initially expected, and so I spent much more time testing for different data-fetching scenarios and designing error widgets than I thought. I would also like to note that I do not plan to stop working on this project when the capstone course is over. I plan to add in the statistics feature soon, and I also plan to still experiment with adding the golf course API further down the line. Ultimately, I would love to see this app put in production so that Strokes Gained Golf users have access to it from the app store.

## **Results**

The Strokes Gained Golf app achieves what I initially set out to do. It offers a mobile application in which golfers can input their lie type and distance to the pin and receive a real-time strokes gained statistic. It also has features that allow for the user to view their statistics from past games, as well as start a new game from courses that have been played and favorited in the past. My software successfully displays a streamlined and intuitive UI to the user and stores and fetches necessary data in a database. It also functions on both iOS and Android devices, which allows for a greater number of users to access its services. Since the back end was built with Cloud Firestore, the app is also highly scalable; I think that, should I eventually put it into production, it would support the user base that Strokes Gained Golf currently has, as well as any growth that the platform experiences in the future.

As was expected, my app also has a few limitations. The app is not currently available in the app store, meaning that Strokes Gained Golf users do not have access to it. Additionally, since I don't know anybody who is a golfer as well as a coder (and currently running the app and emulator requires quite a bit of complicated setup and configuration), I have not had anybody

from the target user base test its features and give comments. Ideally, I would want several golfers to play around with the app its features, and then I would adjust app components according to their feedback. This is especially true since I am not an avid golfer myself, and thus I would appreciate a higher level of insight into what is expected and desired from a golfing app.

I have included a [video](#) that demonstrates the features of the Strokes Gained Golf app.

## Conclusions

In terms of future work, there's still a lot that I hope to do with Strokes Gained Golf. In terms of perfecting features that already exist in the app, I would like for there to be an option for the user to change the number of holes they're playing while they're in the game (switching from 9 to 18 or vice versa) instead of only allowing them to decide when they initially start a new game. Along with this feature, I would also like to allow users to go in after they have completed a game and edit their strokes, which would be helpful if they missed a stroke or added the wrong lie type or distance when they initially inputted it. In the past games section, I would like to add a feature that allows users to sort their games (by date, course name, and time, for example). Therefore, if the user has played many games and is looking for a specific one, it would be easier to locate. Next, I would like to implement an auto-complete feature on the 'New Game' page where the user inputs the course name. Currently, they must manually type the course name, but it would improve user flow and reduce time and effort to start a new game if a list of nearby courses popped up when they clicked the input box and auto filled the box when selected. Lastly, I would like to extend the navigation bar to persist throughout the whole app (it's currently only on the home page). Since flutter builds apps using a widget tree, it is intentionally difficult to create a navigation bar, and I ran out of time to try and implement it.

Initially, I had planned to connect my app's database to the pre-existing database from the website. This turned out to not be feasible because the software engineer that designed the website initially was not available, and it would have taken a lot of meetings and explanations for us to be on the same page. However, I was glad that I had the opportunity to design and implement my own database, especially because I didn't have prior experience with Firebase. In the future, I want to work on connecting my app with the existing database so that Strokes Gained Golf users have access to their information via both the website and the app.

As mentioned previously, there were two features in my initial proposal that I did not get to: implementing the statistics page and integrating a golf course API that automatically tracks the user's location on the course, thus reducing the amount of user input required in the strokes gained calculation process. Since having access to an overview of strokes gained statistics is a big reason for why people are interested in Strokes Gained Golf, I am hoping to implement the statistics page as soon as possible. I also believe that integrating the API would make people much more likely to subscribe to Strokes Gained Golf, as currently one of the biggest complaints about the platform's services is the time and effort it takes to manually input information for each stroke. By integrating an API into the mobile app that automates this process, I think that it would provide even more opportunities for Strokes Gained Golf to expand its user base and grow as a company.

Despite future work to be done, I am very pleased with my final (capstone) product for the Strokes Gained Golf app. I am proud of how the app looks, despite having very minimal graphic designs to use. I think that my design choices, such as using colors from the logo, having

each step of the strokes gained calculation process be on a new screen, and displaying effective error messages resulted in a visually appealing and intuitive app. Additionally, I think that my choices for storing and accessing data using the database do a good job of tailoring the app's features to individual users, and allow for scalability in the future. Working on this app for my capstone project has given me invaluable insight into new tools, has made a much stronger independent coder, and has set up a very strong base for me to grow and improve the Strokes Gained Golf app (and hopefully the company overall) in the near future.