HoleScope: Golf Scorecard and Stats App

Celt Stephenson Li

N00213756

Supervisor: Cyril Connolly

Second Reader: Sue Reardon

Year 4 2022-23

DL836 BSc (Hons) in Creative Computing

Abstract

The aim of this project was to create a golf score tracking app for myself and many other keen golfers to use to keep track of the rounds they play and the stats they accumulate. The game of golf is growing at a huge rate, with the average amount of rounds being played in the UK in 2022 increasing 24% since 2019 (Golf Business News, n.d.). That along with the growing desires of people to have everything on their phone, provide a great opportunity for mobile golf scorecards and stats trackers. I implemented this application over eight weekly sprints using the SCRUM methodology for project management. I used a MERN stack for the applications architecture consisting of MongoDB, Express, React Native and Node.js. Testing was carried out during and after the implementation process and results were positive, showing very little functional issues but with a few UI aspects to look at improving. The app has plenty of room to grow with there being many more formats for golf than just pure single player strokeplay.

Acknowledgements

Special thanks to Cyril Connolly for helping manage my project, giving me advice and keeping me on track.

And a big thank you to all the participants in my interviews, surveys and user testing.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Celt Stephenson Li

Signed Celt S Li

Failure to complete and submit this form may lead to an investigation into your work.

Table of Contents

Table of Figures

1 Introduction

Overall Aim

The goal of this project was to create a Golf Score Tracking App that myself and many other golfers could use. The game of golf is growing rapidly, with the average number of rounds being played in the UK in 2022 increasing 24% since 2019 (Golf Business News, n.d.)., and players are always looking for new ways to enter their rounds and keep track of their stats. Players have historically entered their scores on physical paper scorecards, but they become damaged, lost and forgotten about. This app aims to provide golfers with a new easy way to keep track of their scores, stats and previous rounds all in one place and accessible at the click of a button. I myself am a keen golfer and I am always trying to improve my game. Keeping track of your stats over time lets you know your strengths and weaknesses and gives your practice a focus. Chapter 2 of this document is a literature review comparing different JavaScript Frameworks and how they should be considered when building an application of this type.

Technologies

For technologies I essentially used a MERN (MongoDB, Express, React and Node) stack, however I used React Native instead of just pure React. MongoDB is a NoSQL database which stores my application's data in a JSON like format. I chose MongoDB because of its seamless compatibility with React Native and Express. Express.js is a back end Node.js framework, I used it to build the server for my application. I chose Express because I had previous experience using it and felt it was the most appropriate back-end framework to use. React Native is a JavaScript framework built on the functionality of React, which is one of the most popular JavaScript frameworks in the world used for developing web applications. React Native allows developers to use this framework in conjunction with native platform functionality. It allowed me to write one set of code and deploy it across numerous platforms, these include iOS, Android and Web. The main platform I deployed and tested it on was iOS, I was able to run a local server on my PC and connect to it on my phone via the ExpoGo app. This allowed me to view, utilise and edit the app in real time. Other technologies that could have been considered were Flutter and Swift however for the purpose of my application and the hardware I had access to, React Native was the best choice.

Requirements

I went through a number of steps to gather the functional and non-functional requirements for my application. I first looked at similar applications to mine to gain inspiration and seek any faults or flaws that I could improve upon. Secondly, I conducted interviews and sent out a survey to gather information from prospective users about what they liked or disliked about any golf applications they used, as well as any new ideas they might like to see added to such an app. I then created proto personas of some of the different types of users I may have. This outlined a range of potential users and what their needs and wants may be. I also created a Use Case Diagram to gain a visual understanding of all the different operations users would undertake while using the app. I was then able to develop a full list of all the functional and non-functional requirements for my application.

Design

The first step in this process was to sketch out the various technologies I would be employing and how they would be organized. I researched coding standards and conventions to ensure that my project files were well-structured and organized. I then created an Application Architecture Block

Diagram to get a bird's-eye view of the various aspects of the application I was developing. This clarified the client and server request and response processes, as well as the server's access to the database. I then created an Entity Relationship Diagram to show how the various data tables in my database were structured and how they would interact with one another. The next step was process design, which was aided by the creation of two more diagrams. The first was a Sequence Diagram, which depicted how users would interact with the application's back end. The second diagram was a Flow Chart, which allowed me to visualize the various actions and decisions that users would make as they navigated the application. This aided me in understanding the app's flow and identifying any bottlenecks or issues that might arise. I then moved on to designing user interfaces. Figma was used to create wireframes for my application. I created wireframes for each individual screen to provide myself with a visual goal to strive for while developing the application. I also created a style guide to follow so that I could create a consistent and well-thought-out design for each component and element within the app. This process enabled me to create a visually appealing application with a consistent theme throughout.

Implementation

Implementation was carried out using the SCRUM Development Process, this process was carried out over eight weekly sprints. I would implement a number of items each week and meet with my supervisor to review my progress. We'd then decide on the next set of tasks for the following week. This procedure gave my work a nice structure and set manageable goals for each week.

Testing

The testing process involved conducting functional and user testing on my application. I started this process by using the Black Box Testing technique to run a series of tests on my application's overall functionality. I was only concerned with the actual output matching the expected output in these tests. These tests were successful, demonstrating that I had met all my functional requirements. I then conducted in-person user testing with four prospective users, asking them to complete a series of tasks and recording their results and feedback. The results were positive, with users complimenting the application's usability and design.

Project Management

My project management was aided by the SCRUM methodology where I carried out 8 weekly sprints. I met with my supervisor every Wednesday to discuss and display my progress, we then came up with a new set of items for me to implement for the next week. I used GitHub for my version control, I had repositories for both the front end and back end of my application. I made regular commits along with comments every time I made changes and implemented functionality.

2 Research

2.1 Comparing JavaScript Frameworks and how they Should be Considered when Building an Online Community Website

2.1.1 Introduction

The use of JavaScript frameworks has become widespread in web development, as they provide a structured approach for building interactive and feature-rich websites. When it comes to building online community websites, which often require social networking and communication capabilities, the decision of which JavaScript framework to use is of the utmost importance. In this literature review, I will explore the various JavaScript frameworks available and how they should be considered when building online community websites. I will examine the benefits and drawbacks of using these frameworks and discuss how they can be used to create user-friendly and engaging online community experiences. I will also consider future trends in the field and their potential impact.

2.1.2 JavaScript

JavaScript was released in 1995. It was developed with the intention of incorporating interaction into online sites, it has since grown to be among the most popular programming languages in the world. The front end, or client side of web development makes use of JavaScript, which implies that it runs on the user's computer or device rather than a server. (Jensen, S.H., Møller, A., Thiemann, P. 2009) The ability of JavaScript to alter the Document Object Model (DOM) is one of its key characteristics, the DOM, which is a tree-like structure that reflects the HTML of a web page, by adding, removing, or modifying the components on the page. This allows programmers to construct dynamic and interactive websites with features like picture sliders, drop-down menus, and form validation. (Jeremy Keith, 2005) As an object-oriented language, JavaScript is built on the idea of objects. An object is a group of attributes (variables) and operations (functions) that together reflect a real-world entity. A dog object, for instance, might include attributes like name, age, and breed as well as methods like bark and walk. (Jensen, S.H., Møller, A., Thiemann, P. 2009)

JavaScript has a few main advantages, one of which is that it is quite a simple language to learn, especially for programmers with prior experience. It is also a very adaptable language that can be applied outside of the web browser. (Jeremy Keith, 2005). JavaScript can be used to develop desktop, server-side, and mobile apps. Despite all of its advantages, JavaScript has certain drawbacks. The language is often criticized for not being as quick as some other languages, such C++. For applications that need an elevated level of performance, such video games or scientific simulations, this can be an issue. Furthermore, JavaScript may be vulnerable to security flaws, particularly when it is used to develop intricate applications with a lot of code. To conclude, JavaScript is a strong and popular programming language that has contributed significantly to the growth of the web as we know it today. It is a crucial tool for everyone interested in web development because of its versatility and capacity to add interaction to online pages. (Sanjay Misra and Ferid Cafer, 2012)

2.1.3 JavaScript Frameworks

A JavaScript framework is a collection of pre-written JavaScript code that allows developers to create web applications more quickly and easily. These frameworks provide a structure for developers to adhere to, which can save time and effort when developing complex applications. (Josefin Salomaa, 2020). There are numerous JavaScript frameworks available, each with its own set of features and benefits. React, Angular, and Vue.js are some popular frameworks.

React is a JavaScript library used to create user interfaces. It was created by Facebook and is frequently used to create single-page applications (SPAs). React allows developers to create reusable UI components and improves application performance by reducing the amount of DOM manipulation required. Angular is a JavaScript framework for developing web applications. It was created by Google and includes many features such as two-way data binding, dependency injection, and a powerful template system. Angular is frequently used to create large, complex applications. Vue.js is a lightweight JavaScript framework that is intended to be simple to learn and implement. It has a simple template syntax and a reactive component model, making it an excellent choice for creating interactive user interfaces. (Elar Saks, 2019)

One of the primary advantages of utilizing a JavaScript framework is that it can aid in the performance of your web application. Frameworks give your code structure and can help you write more efficient and optimized code. They also frequently include built-in tools and features that can assist you in building and testing your application more quickly and easily. JavaScript frameworks, in addition to improving performance, can make it easier to develop web applications. They frequently provide a set of conventions and best practices that can aid in the creation of cleaner, more maintainable code. This can save you time and effort in the long run when developing and maintaining your application. (Pano, A., Graziotin, D & Abrahamsson, P. 2018). Overall, JavaScript frameworks can be a useful tool for web application developers. They can help improve performance, make development and maintenance easier, and provide a framework for writing efficient and well-organized code. A JavaScript framework can be a useful addition to your toolkit whether you are an experienced developer or just starting out. (Joesfin Salomaa, 2020)

2.1.4 Comparing Different JavaScript Frameworks

There are numerous JavaScript frameworks available, each with its own set of features and capabilities. We will compare several popular JavaScript frameworks in this section to help developers understand the key differences between them.

React is a popular framework for creating user interfaces. It makes use of a virtual DOM (Document Object Model) to improve performance and make complex user interfaces easier to build and update. React is well-known for its ease of use and flexibility, and it is a popular choice among developers who are creating applications with complex user interfaces. Angular is a comprehensive framework that can be used to create both simple and complex applications. It includes a number of powerful features, such as two-way data binding and dependency injection, which help developers build and maintain applications more easily. Angular is well-known for its extensive feature set and strong emphasis on developer productivity. Vue.js is a lightweight framework that is intended to be simple to learn and apply.

It is especially popular for creating single-page applications, and it places a strong emphasis on performance. Vue.js is well-known for its ease of use and flexibility, making it a popular choice for developers creating applications with a strong emphasis on user experience. (Elar Saks, 2019) Ember.js is a full-featured framework for developing complex applications. It has a robust feature set, including a powerful router and a templating system, and it is well-known for its strong emphasis on developer productivity. Ember.js is a good choice for developers who need to create complex applications with a lot of features and functionality.Backbone.js is a lightweight framework for creating single-page applications. It is well-known for its simplicity and flexibility, and it is a popular choice among developers looking to create applications that are simple to maintain and update over time. (S. Delcev and D. Draskovic, 2018).

To sum up, there are a variety of JavaScript frameworks available, each with an own set of features and functionalities. While Angular is a complete framework ideal for creating both simple and complex apps, React is an excellent option for developers building applications with complex user interfaces. While Ember.js is a fully featured framework appropriate for constructing complicated apps, Vue.js is a lightweight framework that is especially popular for building single-page applications. Finally, Backbone.js is a popular framework for creating single-page apps that are simple to update and manage. The optimal JavaScript framework for a given project will ultimately depend on the demands and objectives of that project. (S. Delcev and D. Draskovic, 2018).

2.1.5 Website Development

The process of building and managing a website is called website development. It requires a variety of tasks, including as planning, designing, developing, testing, and deploying. A team of designers, developers, and content producers often work together during the website development process because it may be a challenging process, especially for larger and more feature-rich websites. Planning is the first stage of website creation. This include determining the website's objectives and aims, as well as its target market. It also entails creating a project roadmap and deciding what features and content will be featured on the website. (M.J Taylor, J McWilliam, H Forsyth, S Wade, 2002). Design is the following phase. This include developing the website's visual design, including the colour scheme, font selections, and general style. When creating a website, it is crucial to keep in mind the target audience and the site's objectives. The layout should be aesthetically pleasing and simple to use for the target audience. The website's development comes after the design phase is finished. This include creating the HTML, CSS, and JavaScript code that will run the website. Along with testing and debugging the code to make sure it is working properly; development may also involve integrating the website with databases or other back-end services. (Flavian, C., Gurrea, R. and Orús, C., 2009)

It is crucial to test the website after it has been created to make sure it is operating effectively and that it satisfies the project's needs and objectives. To make sure the website is responsive and functions correctly, it may be necessary to test it across a variety of platforms and browsers. Deployment should take place when the website has undergone testing and is prepared to launch. This entails publishing the website online and uploading it to a server. Due to the potential ramifications of downtime or security breaches for both the website and its visitors, it is crucial to keep the website secure and up to date. In conclusion, creating a website requires a variety of steps, such as planning, designing, developing, testing, and deploying. A group of designers, developers, and content producers must work together on it, and it is crucial that they keep the website's target audience and objectives in mind as they go. A website must be successfully planned, tested, and maintained in order to meet the needs of its users. (Davies, A., Mueller, J. 2020)

2.1.6 Online Communities

Online communities are gatherings of people who use the internet to discuss shared passions, concepts, and experiences. They can be found on many different websites and online communities, including social media and forums. From interests and hobbies to social and political issues, online communities can be centred on a wide range of subjects. Connecting with others who have similar interests or life experiences is one of the key advantages of online communities. For people who might not have access to a local community or who might feel lonely in their offline lives, this might be extremely helpful. (Oliver K. Burmeister, 2010) Online communities can offer a sense of community and support that may be hard to find in other settings.

Additionally, information and resources can be found via online communities. The community can act as a collective knowledge and resource pool by allowing members to share their knowledge and experience. This might be especially helpful for people who are looking for information on a certain subject or who need advice or direction. Online communities may be a source of amusement and creativity in addition to connecting people and giving them access to information. Many online groups are dedicated to creative activities like writing, art, or music and offer a forum for members to share their work and get criticism from other users. (Robert Plant, 2004)

Online communities do have some difficulties, though. The possibility of bullying or harassment occurring online is one of the key worries. To prevent this kind of behaviour, online communities should have clear rules and policies in place. Members should also be careful how they engage with one another. The potential for the propagation of false information is another difficulty. Online communities should promote critical thinking, the utilization of trustworthy sources, and users being cautious with the information they post. In general, online communities can be a great tool for anyone looking for connections, knowledge, and creativity. Community members must, however, be aware of their relationships with others and sceptical of the information they come across. Online communities can offer their members a significant source of support and connection by promoting a positive and encouraging environment. (Jacob Amendie, 2015)

2.1.7 Characteristics of an Online Community Website

Websites that serve as online communities enable users to connect and exchange interests, concepts, and experiences. Depending on their goal and intended audience, these websites can have a wide range of qualities. Online community websites often include the following characteristics:

A concentration on a specific topic or interest: Online community websites are frequently built around a specific interest, hobby, or issue. This can range from a narrow interest, like knitting or gaming, to a more general subject, like parenting or social justice. An online community website should have a strong sense of community, which is one of its key features. Members may engage in interpersonal communication with one another through forums, comments, or other channels. An online community website should encourage a feeling of support and community among its users. Online community websites frequently include a range of content, such as articles, blog entries, forums, and user-generated content. The website itself may supply this content, or website visitors may submit it. User profiles: Users can post information about themselves and connect with other users on many online community platforms. These profiles may contain facts on the user's name, address, interests, and other personal information. Moderation: To maintain a secure and welcoming community, online community websites frequently employ moderation. To address inappropriate activity, this may entail the employment of moderators, rules and guidelines, and reporting systems. Mobile friendliness: It is critical for online community websites to be mobile-friendly as an increasing number of people access the internet via smartphones. This can entail having a responsive design that adapts to various screen sizes and devices, as well as a mobile app or a website that has been optimized for mobile devices. Privacy and security: Online community websites should put their users' privacy and security first. This can entail using secure servers and encryption, as well as putting in place explicit standards to safeguard members' privacy. (Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong, 2007)

Online community websites serve as venues for bringing individuals together to discuss common interests, concepts, and experiences. A concentration on a specific subject or interest, a sense of community, a variety of content, user profiles, moderation, mobile accessibility, and security and privacy are common features. These qualities contribute to the development of a community that is uplifting and encouraging for its residents. (Hsiu-Fen Lin and Gwo-Gwang Lee, 2007)

2.1.8 JavaScript Framework Choice when Building an Online Community Website

The correct JavaScript framework must be used while creating an online community website for it to be dependable, scalable, and simple to manage. JavaScript frameworks are pre-written code libraries that offer a framework for web application developers to create. There are a variety of JavaScript frameworks available, each with an own set of features and functionalities. The size and complexity of the website should be considered while selecting a JavaScript framework for an online community website. A lightweight framework like Vue.js or Backbone.js may be adequate for smaller and basic websites. These frameworks are suitable for creating single-page applications and are simple to learn and use. A full-featured framework like Angular or Ember.js would be a preferable option for bigger and more complicated online community websites. These frameworks offer a substantial selection of capabilities and are appropriate for creating more intricate applications. They might have a steeper learning curve, but by offering a collection of reusable parts and functions, they can ultimately save time and effort. (S. Delcev and D. Draskovic, 2018).

The requirements and objectives of the website for the online community are a crucial consideration to consider when selecting a JavaScript framework. For instance, a framework with strong support for data binding and real-time updates, such as Angular or React, may be a smart choice if the website is centred on user-generated content and community involvement. A framework like Vue.js or React may be a better option if the website is focused on performance and scalability. When selecting a JavaScript framework, the development team should also be considered, even if it is not the framework with the most features and capabilities, it could be more effective to use it if the team has existing expertise with it. In conclusion, there are a variety of things to consider while selecting a JavaScript framework for an online community website. It is crucial to consider the size and complexity of the website, the needs and objectives of the project, and the knowledge and expertise of the development team. In order for developers to create a dependable, scalable, and easy to maintain online community website they must carefully weigh their alternatives and selecting the best framework. (Robert Satrom, 2018)

3 Requirements

3.1 Introduction

In this section I will search for all the features and abilities my App will have. I will begin the process by looking at similar applications to mine to see aspects which I like and ones that I do not. This will give me a good base of features for me to implement in my project.

I will then ask users in a variety of ways what they would like the App to be able to do. I will first conduct a series of interviews with potential users to understand the features that they want to see or would like to see improved from similar applications. I will also create a questionnaire asking users for ideas for features for the application.

After gathering my data, I will then use this information to outline both the functional and non-functional requirements for my project. These requirements will structure the work I will complete over the implementation process.

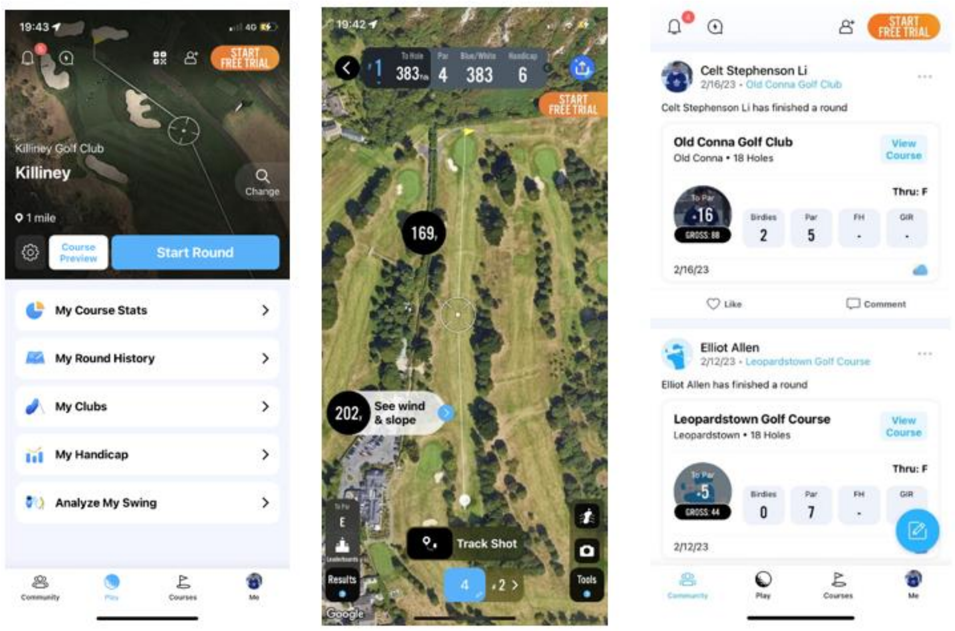
3.2 Requirements gathering

3.2.1 Similar applications

I looked at two main similar applications when researching apps out there already. The first one being:

18Birdies

This is an app I use myself regularly when playing golf. It provides GPS tracking, so you know how far away everything is on the course. It also allows you to track your score while you are playing and stores your scores and statistics so you can review them at any time. There is a free version and a paid version I use the free version which provides score tracking and yardages. The paid version provides more in depth statistics and accounts for wind conditions and slope when giving you your yardages.



Celt Stephenson Li
2/16/23 • Old Conna Golf Club
Celt Stephenson Li has finished a round
Old Conna Golf Club
Old Conna • 18 Holes
View Course
Thru: F
To Par +16 GROSS: 88
Birdies 2
Par 5
FH -
GIR -
2/16/23
Like
Comment

Elliot Allen
2/12/23 • Leopardstown Golf Course
Elliot Allen has finished a round
Leopardstown Golf Course
Leopardstown • 18 Holes
View Course
Thru: F
To Par +5 GROSS: 44
Birdies 0
Par 7
FH -
GIR
2/12/23

Figure 1 – 18 Birdies screenshots

Shown in Figure 1 above on the left is the Play tab on 18Birdies. This is where you begin a round of golf at whichever course you want, the closest geographically pops up first which is very useful, but you can change to an alternative course. You can also view your previous rounds and stats from that course. Shown in Figure 1 above in the middle is the Hole screen. Here you can see an aerial view of the hole you are playing. It displays yardages from your exact location on the golf course making it an extremely useful tool. When you complete the hole, you can then enter your score and extra information about the hole and move on to the next. There is also a social aspect to the App as you can add friends, chat to them and see the rounds they play as well. Shown in Figure 1 on the right side is the community tab on the App, this tab displays a feed of all the activity you and your friends are getting up to.

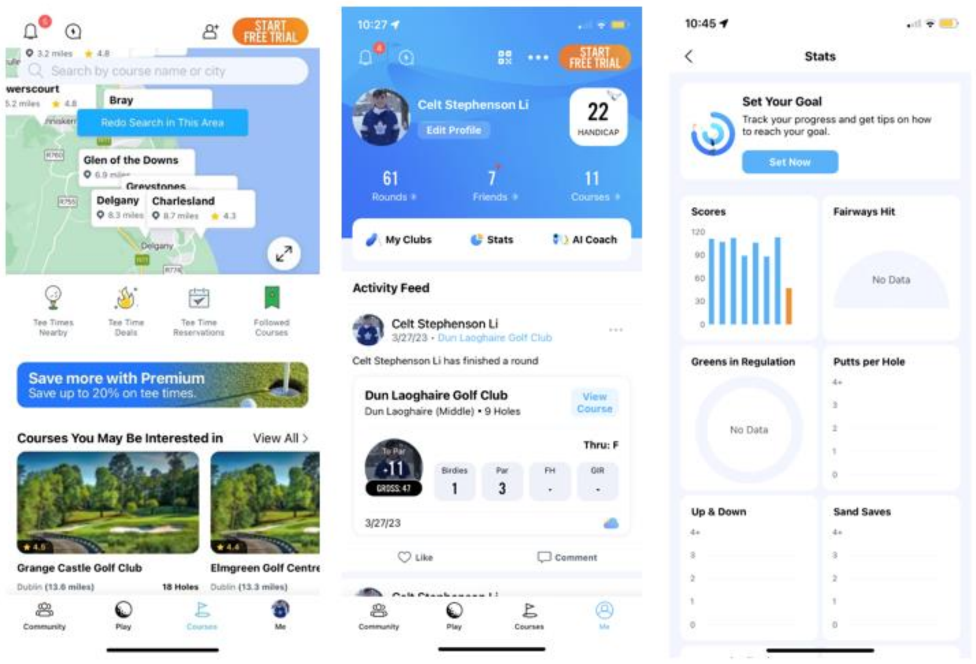


Figure 2 – 18Birdies screenshots continued

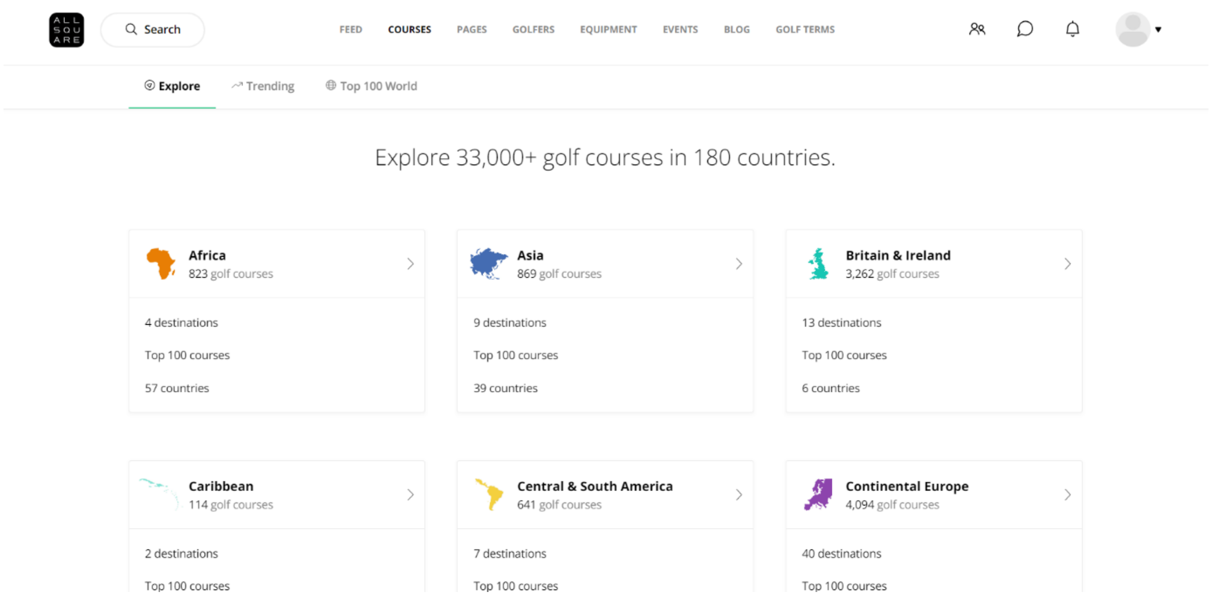
Figure 2 above on the left side shows the Courses tab in 18Birdies, here you can search for different courses using the search bar or using the interactive map. You can also book tee times at different clubs in your area and discover new clubs that are recommended to you. Figure 2 above in the middle shows the Me tab in 18Birdies which has a record of all the rounds you have played in your profile. You can also keep track of the golf clubs you have in your bag and how far you hit the ball with each club so it can make suggestions to you while you are on the course. You can also view your overall statistics from all the rounds of golf you have played. Figure 2 above on the right side shows the Stats screen. Here is a list of different statistics that you can track. I mainly just track my overall score, but you track a large array of different areas including fairways hit, greens in regulation and putts per hole. This information is very useful to have when you are trying to improve your game, so you know what you need to work on.

Overall 18Birdies is a great application that I use regularly, and I want to take a lot of aspects from it to use in my project, mainly the score recording and stats tracking. The free version of this application provides more than enough features but the paid version really takes it to a new level. It would be great to add as many features as I can but there is too many and it’s better to focus on a few aspects that I find the most important.

The next similar application I looked at was:

<u>All Square Golf</u>

This was the first application I found when I was researching the concept of a golf social media app. It differs from 18Birdies as it is more of a web application than a mobile application. It does have a mobile app, but it is better suited to the web. It is less of an app that you use while you are playing golf as it is one that you would use when planning where you are going to play. It also features a LinkedIn like aspect where you can set up your profile and connect with golfers all around the world. Even professional golfers have accounts that you can connect with you can see what equipment they are using and where they are playing all around the world.

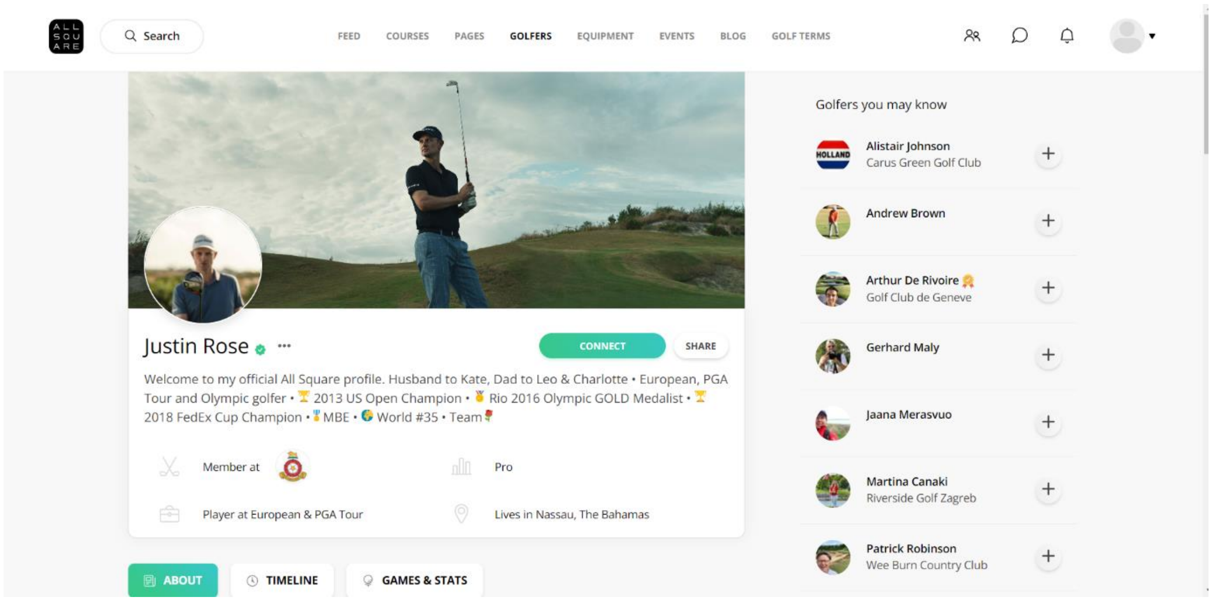


Explore 33,000+ golf courses in 180 countries.

Africa 823 golf courses	**Asia** 869 golf courses	**Britain & Ireland** 3,262 golf courses
4 destinations	9 destinations	13 destinations
Top 100 courses	Top 100 courses	Top 100 courses
57 countries	39 countries	6 countries
Caribbean 114 golf courses	**Central & South America** 641 golf courses	**Continental Europe** 4,094 golf courses
2 destinations	7 destinations	40 destinations
Top 100 courses	Top 100 courses	Top 100 courses

Figure 3 - All Square Golf Courses Page

Figure 3 above shows the Courses page, it lets you discover golf courses all around the world and rates the top one hundred courses in most destinations you can think of. There is also a trending page where you can see what golf courses are the most popular at the moment by geographical area.



Justin Rose

Welcome to my official All Square profile. Husband to Kate, Dad to Leo & Charlotte • European, PGA Tour and Olympic golfer • 2013 US Open Champion • Rio 2016 Olympic GOLD Medalist • 2018 FedEx Cup Champion • MBE • World #35 • Team

Member at

Pro

Player at European & PGA Tour

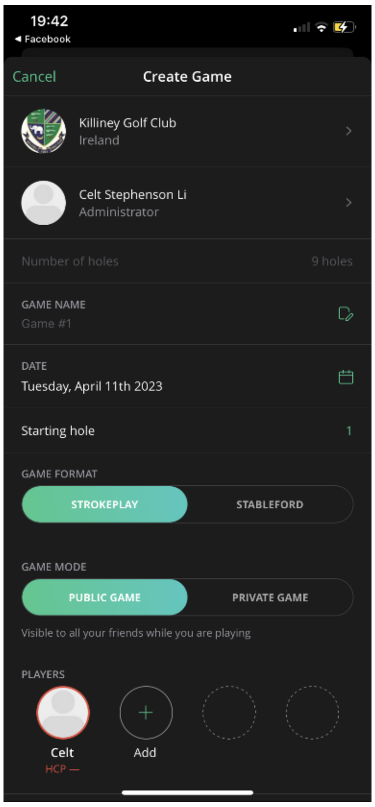
Lives in Nassau, The Bahamas

Golfers you may know

- Alistair Johnson — Carus Green Golf Club
- Andrew Brown
- Arthur De Rivoire — Golf Club de Geneve
- Gerhard Maly
- Jaana Merasvuo
- Martina Canaki — Riverside Golf Zagreb
- Patrick Robinson — Wee Burn Country Club

Figure 4 - Professional Golfers Page on All Square Golf

You can see in Figure 4 above the social aspect of the application. Justin Rose who is one of the top golfers in the world has his own account on the site. You can connect with him, and he actually posts regularly to his timeline updates, pictures and videos showing his preparation for big tournaments including The Masters. This is quite an interesting aspect of the site but ultimately you can just follow most players on Instagram or other social media apps and get the same kind of content. So, it is not something I think I need to implement in my application.


19:42
◂ Facebook

Cancel Create Game

Killiney Golf Club
Ireland

Celt Stephenson Li
Administrator

Number of holes 9 holes

GAME NAME
Game #1

DATE
Tuesday, April 11th 2023

Starting hole 1

GAME FORMAT
STROKEPLAY STABLEFORD

GAME MODE
PUBLIC GAME PRIVATE GAME

Visible to all your friends while you are playing

PLAYERS
Celt
HCP —
Add

Figure 5 – Play Screen on All Square Golf App

Above here in Figure 5 is the Play Screen on the mobile application. This aspect of the app does not really work at all I have not even been able to test it out as I cannot even get a round to start on the app. This feature would be the only real reason to have it on mobile and because it does not work, I find the mobile app to be quite pointless.

Overall, All Square Golf is a useful tool for keen golfers looking to discover courses all over the world and connect with other golfers. Ultimately my project will focus more on the playing and stats tracking aspects of golf applications, will leave out a large social network and course recommendations all over the world.

3.2.2 Interviews

I conducted interviews with 3 potential users who were all keen golfers that used score tracking apps. I asked them questions about what they enjoyed and disliked about the apps they were using as well as features on the app they used the most and the least. I got useful information from all 3 interviewees. All were satisfied with the apps they are using but had some suggestions to cater for themselves personally. Two of the participants would like their app to be a bit simpler and straight to the point there was a lot of unnecessary bells and whistles. One remarked that they liked the number of statistics they were able to track and that it really helped their progression. When I asked the interviewees if they liked the social aspect of their apps they all had similar things to say. The general sentiment was that while they like that they can add their friends on the app and see what they are doing, they don't think it is a very important aspect of the app for them and they don't pay much attention to it.

I concluded I don't particularly have to reinvent the wheel, I just need to tweak and alter certain aspects of the existing apps in order to give the user what they want. Overall, this was a very worthwhile activity and gave me a nice look at my project from an outside perspective.

3.2.3 Survey

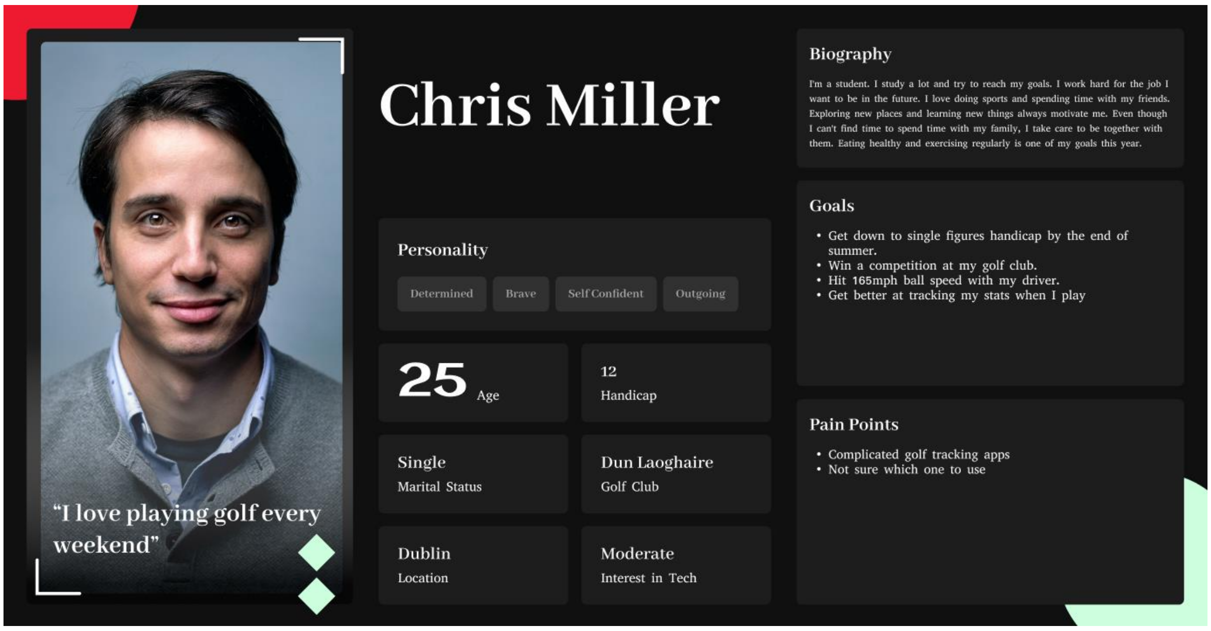
I created a questionnaire for my friends asking them an array of questions about their current golf apps. These questions were a mixture of yes or no and short response answers. I have received 5 responses to this point and I can see trends revealing themselves in people's answers. Most people are looking for a very simple interface that is straight to the point in their eyes, the Play tab is the most important tab for most users and the tab they want to get to the quickest and easiest. The minority of users are looking to track stats other than purely their score, i.e. fairways hit, putts etc. I suspected this because I am in the same category. They did portray interest in tracking more stats however, if it was made to be quicker and simpler.

In conclusion I got some good data from this questionnaire. I am going to use the thoughts and feelings from my friends to make decisions on what features I feel are the most important to implement.

3.3 Requirements modelling

3.3.1 Personas

I created two different Proto Personas to understand what different prospective users would be looking for in a golf score tracking app.



“I love playing golf every weekend”

Chris Miller

Personality

Determined | Brave | Self Confident | Outgoing

25 Age	12 Handicap
Single Marital Status	Dun Laoghaire Golf Club
Dublin Location	Moderate Interest in Tech

Biography

I'm a student. I study a lot and try to reach my goals. I work hard for the job I want to be in the future. I love doing sports and spending time with my friends. Exploring new places and learning new things always motivate me. Even though I can't find time to spend time with my family, I take care to be together with them. Eating healthy and exercising regularly is one of my goals this year.

Goals

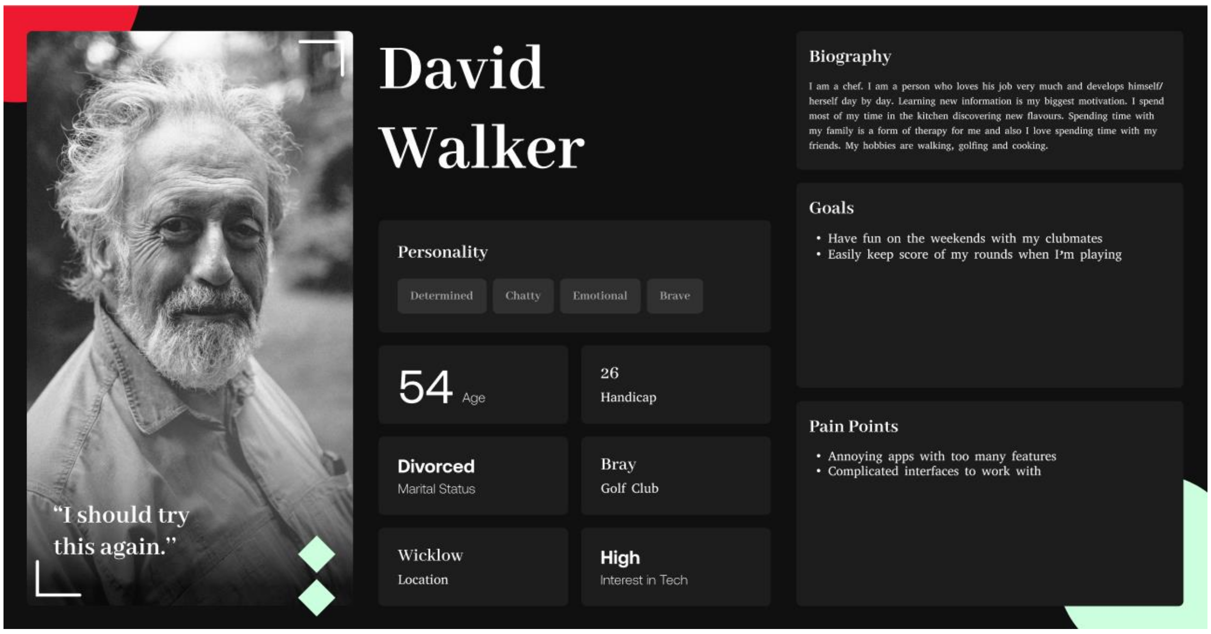
- Get down to single figures handicap by the end of summer.
- Win a competition at my golf club.
- Hit 165mph ball speed with my driver.
- Get better at tracking my stats when I play

Pain Points

- Complicated golf tracking apps
- Not sure which one to use

Figure 6 - Proto Persona of avid Golfer

Shown in Figure 6 above is a proto persona for an avid golfer looking to track his stats intently as he plays and improve his game as much as he can. This person will be looking for an app that provides a lot of functionality to provide him information on his game that can help him improve.



“I should try this again.”

David Walker

Personality

Determined | Chatty | Emotional | Brave

54 Age	26 Handicap
Divorced Marital Status	Bray Golf Club
Wicklow Location	High Interest in Tech

Biography

I am a chef. I am a person who loves his job very much and develops himself/ herself day by day. Learning new information is my biggest motivation. I spend most of my time in the kitchen discovering new flavours. Spending time with my family is a form of therapy for me and also I love spending time with my friends. My hobbies are walking, golfing and cooking.

Goals

- Have fun on the weekends with my clubmates
- Easily keep score of my rounds when I’m playing

Pain Points

- Annoying apps with too many features
- Complicated interfaces to work with

Figure 7 - Proto Person of casual user

Figure 7 above shows a Proto Persona of a casual golfer who simply plays for fun on the weekends with his friends. He will not be looking for a lot of complicated features. He is more likely to just purely track his score as he is playing with his friends.

3.3.2 Functional requirements

1. A user will be able to sign up and log in.
2. Appropriate data from the back end will be able to be viewed on the front end.
3. A user will be able to search for their desired course.
4. A user will be able to start a round at that golf course.
5. A user will be able to enter their scores and additional information about each hole of their round.
6. A user will be able to submit this round and it will be stored in the backend.
7. A user will be able to view previous rounds they have played and their overall stats from rounds they have played.

3.3.3 Non-functional requirements

1. The UI should be clean, simple and attractive.
2. The application should perform tasks quickly and smoothly.
3. The application should be suitable for both iOS and Android.

3.3.4 Use Case Diagrams

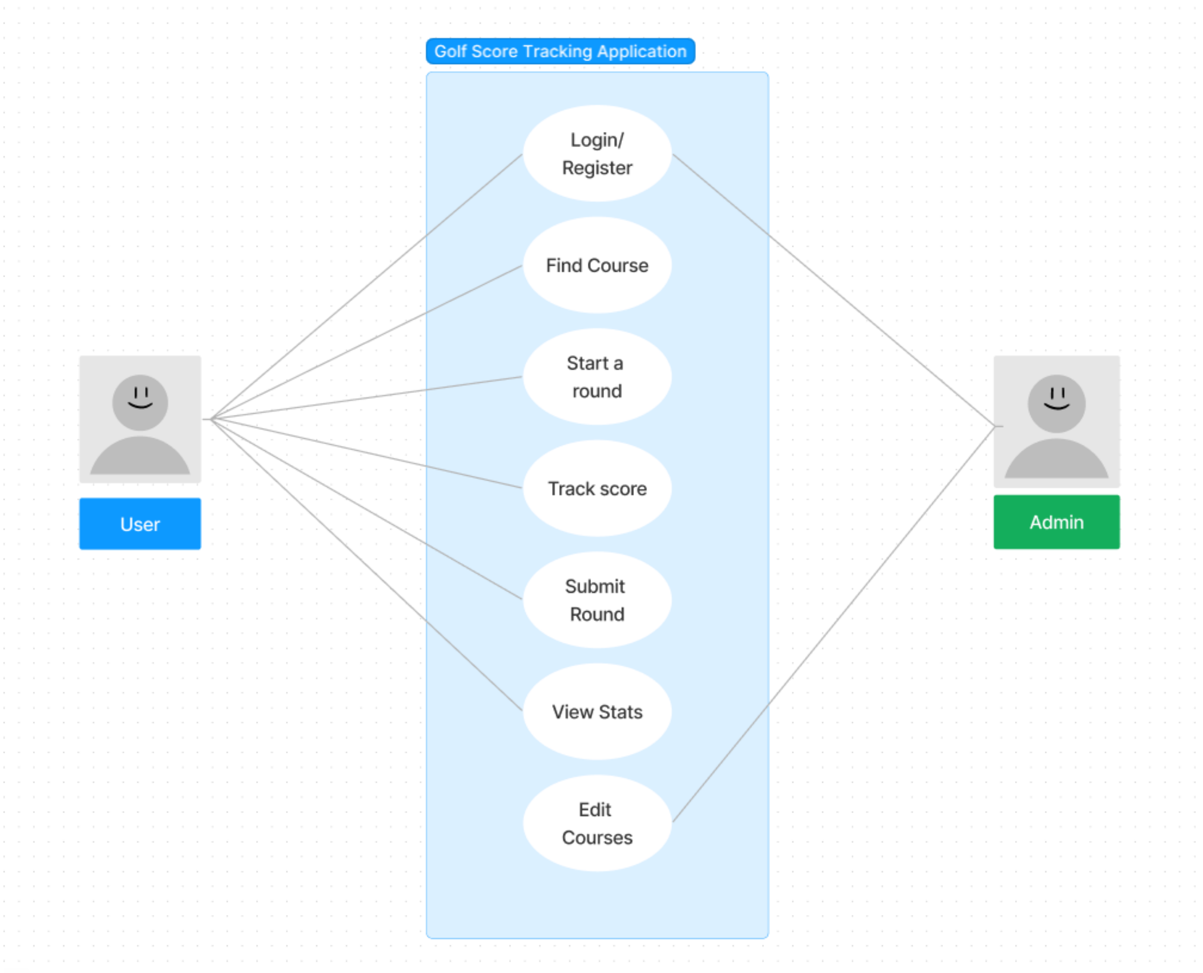


Figure 8 - Use Case Diagram for golf score tracking application

3.4 Feasibility

For the front end of my project, I will be using React Native, this is an open-source JavaScript framework used for developing applications on platforms such as iOS, macOS, Android and Windows. It enables developers to use the React framework along with native platform capabilities. For the back end of my project, I will be using Express.js, which is a back-end web application used for building REST APIs with Node.js. There will be no issue combing these two frameworks for creating my application. I will create my database using MongoDB, this is a no SQL database that uses a JSON-like format to store documents.

For project management I will be using GitHub for my code I will have two repositories, one for my front end and one for my backend. Committing code regularly to keep a record of work I am completing. Finally, I will be using Figma to create wireframes and any other design or architecture related diagrams. All of these technologies should have no issue working together.

3.5 Conclusion

Overall, in this chapter I looked at our project and decided upon what I needed to do. I spent time looking at similar applications in order to gain a good understanding of what already exists so that I had a good base for my project. I then came up with ways I could improve upon or use aspects from these apps. This was an important step to take to give me a good starting point. I then conducted interviews and a survey to gain a good understanding of how people feel about the current golf score tracking apps that are available and what they would like to see changed or improved. I got valuable information which really helped give me a focus on what I will be attempting to create.

I then modelled personas and a use case diagram to help visualize who will be using my app and how they will be using it. It is important for me to see our project from the user's perspective so that I don't lose sight of what my end goal is, which is to create a very user-friendly app experience which people would recommend to their friends and family. Having collected all the aforementioned information, I decided upon both the functional and non-functional requirements. I just listed everything I need my app to be able to do and how I want my app to behave. This helped give me a nice checklist for my project. I finally decided on all the technologies I am going to use for this project and explored the feasibility of my project working with these different technologies to create the app that I desire to make. In conclusion this chapter documented a very important first step in creating our project as I now have a good idea of all I need to create it and what people want to see in our app.

4 Design

4.1 Introduction

The application for this project is HoleScope, a golf score tracking app where users can sign up, log in find the golf course they are playing and start a round of golf. They will be able to enter information about each hole they play including their score, whether they the fairway with their drive, whether they hit the green in regulation and how many putts they took on the green. They will be able to submit these rounds and they will be stored in the database. They will then be able to view their stats from each round they play and an overall look at how they are performing over time.

4.2 Program Design

4.2.1 Technologies

The technologies being used to create this application are:

- React Native

  React Native is a mobile application development framework that allows developers to create code once and deliver it across several platforms, including iOS and Android. It is built on React, a JavaScript library for generating user interfaces, and approaches mobile app development similarly. React Native allows developers to construct mobile applications with a single codebase, increasing efficiency and cost-effectiveness.

- Node.js

  Node.js is an open-source server-side technology that allows developers to create high-performance and scalable online applications. It is built on the V8 JavaScript engine, which is used by the Google Chrome browser, and offers an event-driven, non-blocking I/O architecture, making it perfect for developing real-time, data-intensive applications. Node.js includes a broad ecosystem of libraries, frameworks, and tools to assist developers in creating robust and efficient applications, including popular frameworks like as Express.js, Nest.js, and Hapi.js. It also has advantages like easy scalability, fast performance, and many modules, making it a top choice for developers wishing to construct JavaScript backend apps.

- Express

  Express.js is a Node.js web application framework that simplifies the development of web apps and APIs. It includes capabilities like routing, middleware, template engines, and error handling that make it simple to build powerful and scalable web applications. Because

Express.js is lightweight and adaptable, it is simple to combine with various third-party libraries and frameworks. It also has a big and active community that helps with its creation, upkeep, and support. Express.js is a popular framework for developing RESTful APIs, online apps, and server-side applications that require a simple and straightforward framework.

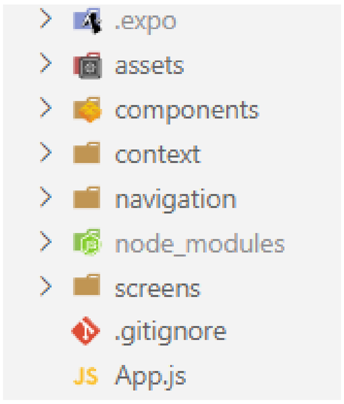
- MongoDB

  MongoDB is a NoSQL database made to manage unstructured data in a scalable and adaptable way. MongoDB uses a document-based data model as opposed to conventional relational databases, making it possible to store and retrieve complex data structures quickly and easily. It is a popular option for contemporary online and mobile apps because of its reputation for handling high volume and high velocity data with low latency. For developers searching for a reliable and adaptable database solution, MongoDB is a fantastic option because of its strong community support, abundance of information, and tools.

I am combining these technologies to essentially create a MERN stack which is a popular web development stack except instead of using pure React, I am using React Native. This stack has a number of key advantages including its flexibility, ease of use and speed of development. Due to Node.js's non-blocking I/O capabilities and MongoDB's potent data modelling skills, it also provides excellent performance and scalability. The MERN stack is a fantastic option for developing apps as it is open source and has a huge and active community of developers contributing to its development. This also aids a lot of troubleshooting processes as there are resources online to help fix most issues into which I might run. I had initially investigated the feasibility of writing my code using Swift on Xcode in order to create a pure iOS app however I was unable to do so as I only have a Windows PC and you need a mac to run Xcode. Because of this React Native was a great alternative.

4.2.2 Structure of React Native and Express

React Native



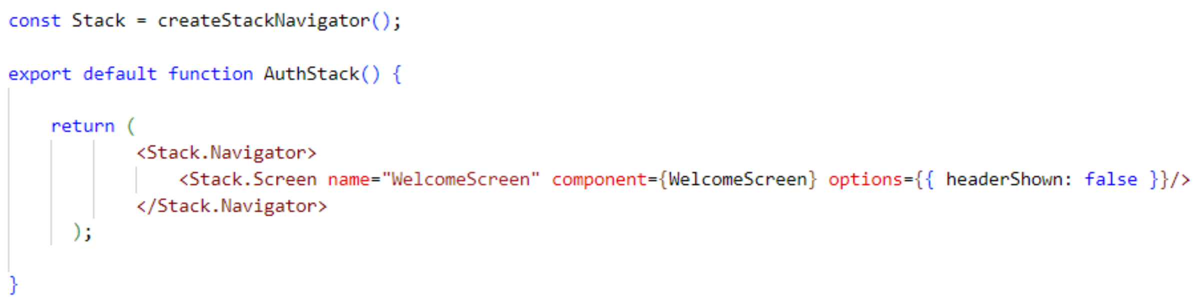
```
> .expo
> assets
> components
> context
> navigation
> node_modules
> screens
  .gitignore
  App.js
```

Figure 9 - Folder Structure of Front End

Shown in Figure 9 above is the folder structure of my React Native project in Visual Studio Code. I split my files into folders for cleanliness and clarity. The components folder contains all the individual components used to make up the application. These components can be reused anywhere in the application that I need.

The context folder contains one file called AuthContext.js, this file contains the login in and sign-up functionality and provides the rest of the files in the application with values such as the userToken and functions such as getUser that can be used at any time utilising the useContext hook from React.

The navigation folder contains three files AppNav.js, AppStack.js and AuthStack.js. These files decide which screens are viewable by the user depending on whether they have been authenticated or not.



```
const Stack = createStackNavigator();

export default function AuthStack() {

    return (
            <Stack.Navigator>
                <Stack.Screen name="WelcomeScreen" component={WelcomeScreen} options={{ headerShown: false }}/>
            </Stack.Navigator>
    );

}
```

Figure 10 – AuthStack.js file

Shown in Figure 10 above is the Screen Stack that the user sees when they are unauthenticated. This is just the WelcomeScreen component that contains log in and sign-up options.

```
const Stack = createStackNavigator();

export default function AppStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="BottomTabNavigator"
        component={BottomTabNavigator}
        options={{ headerShown: false }}
      />
      <Stack.Screen name="CoursesScreen" component={CoursesScreen} />
      <Stack.Screen name="HomeScreen" component={HomeScreen} />
      <Stack.Screen name="MeScreen" component={MeScreen} />
      <Stack.Screen name="PlayScreen" component={PlayScreen} />
      <Stack.Screen name="ShowCourseScreen" component={ShowCourseScreen} />
      <Stack.Screen name="HoleScreen" component={HoleScreen} />
      <Stack.Screen name="LoginForm" component={LoginForm} />
      <Stack.Screen name="StatsScreen" component={StatsScreen} />
      <Stack.Screen name="SingleStatScreen" component={SingleStatScreen} />
    </Stack.Navigator>
  );
}
```

Figure 11 - AppStack.js file

Shown in Figure 11 above is the AppStack.js file which contains the Screen Stack for users that are authenticated. This is the Stack that users will be navigating around when they are using the app, it contains all the main Screen components.

```
function AppNav() {
  const { isLoading, userToken } = useContext(AuthContext);

  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <ActivityIndicator size={"large"} />
      </View>
    );
  }

  return (
    <NativeBaseProvider theme={theme}>
      <NavigationContainer>
        {userToken !== null ? <AppStack /> : <AuthStack />}
      </NavigationContainer>
    </NativeBaseProvider>
  );
}
```

Figure 12 – AppNav.js file

Shown in Figure 12 above is the AppNav.js file. This file acts as a usual App.js file would and decides which Stack file to display based on whether the userToken value is null or not. The AppNav is then the only component passed into the actual App.js file.

The Screens folder contains the files for all the individual Screens in my project. These files are comprised of functions and components that make up the UI for my application.

Express

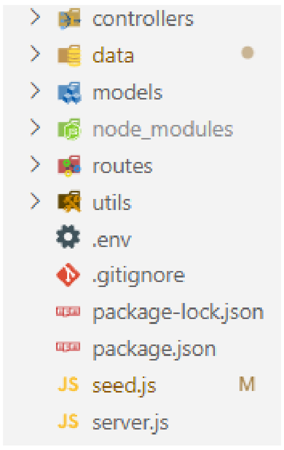


Figure 13 – Structure of Express folders

Shown above in Figure 13 is the folder structure of my Express application in Visual Studio Code. The controllers folder contains 3 files, auth_controller.js, course_controller.js and user_controller.js. The files contain the functionality for each aspect of the application. The data folder contains the JSON file of golf courses that I seed my database with. The models folder contains the user_schema and course_schema files which determine the types of values that are stored in the user and course tables in the database.

```
const userSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'Name field is required'],
            trim: true
        },
        email: {
            type: String,
            required: [true, 'Email field is required'],
            unique: true,
            lowercase: true,
            trim: true
        },
        password: {
            type: String,
            required: [true, 'Password field is required'],
        },
        favourite_courses: {
            type: [String]
        },
        played_courses: {
            type: [Object]
        },
        stats: {
            type: Object
        }

    },
    { timestamps: true }
);
```

Figure 14 - user_schema.js file

Shown in Figure 14 above is the user_schema.js file this contains Mongoose data types which the MongoDB database uses to store the values.

The routes folder then contains two files, one for the users endpoints in the API call and one for the courses endpoints. These decide which functions to run when each endpoint is called by the front end.

The utils folder contains the db.js file which is the file that connects the application to the database using Mongoose. The seed.js file seeds my database with all the courses in my list_courses.json file. Finally, the server.js file contains all of the Express middleware used to run the back end of my application.

4.2.3 Application architecture

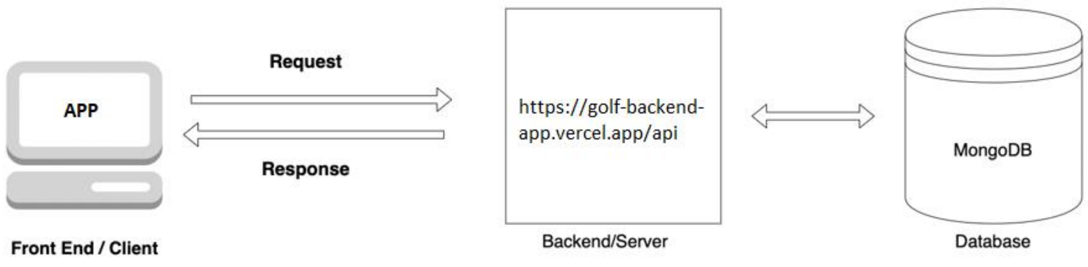


Figure 15 – Block diagram of Application Architecture

A React Native, Express, and MongoDB app can be thought of as having a full-stack JavaScript architecture. Express provides the server-side architecture and manages HTTP requests and answers, while React Native provides the application's front-end components and user experience. MongoDB handles the application's database storage and retrieval. The client-side React Native application communicates with the server-side Express API via HTTP requests, which the server processes and responds to. The server-side Express application connects with the MongoDB database to save and retrieve data.

A scalable and adaptable framework for creating mobile applications that need back-end assistance is provided by this architecture. Additionally, the uniform development environment offered by JavaScript use throughout the stack makes it simple for developers to work on the application's front end and back end. In conclusion, the app architecture of React Native, Express, and MongoDB offers a cutting-edge and potent method for creating mobile applications.

4.2.4 Database design

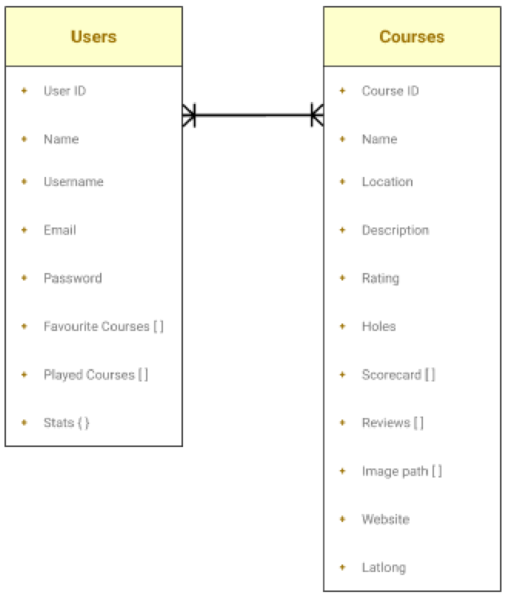


Figure 16 – ERD of my application

In Figure 16 above you can see the Entity Relationship Diagram for my application. This outlines how my database is structured. I have two main tables in my database, the users table and the courses table. The user table is used to store data of each individual user who signs up. They are given a unique ID which is generated by MongoDB, they are also set up with some empty values for themselves that will be filled out as they use the application and submit rounds. The courses table provides all the information on the golf courses that are seeded into the database. Similarly, a unique ID is created for each individual course. Users can interact with one or many courses and similarly courses can be interacted with by one or many users.

4.2.5 Process design

Process design diagrams are tools for organizing and carrying out the software application development process. They help the developer better comprehend the process flow and spot possible areas for optimization with the use of these diagrams, which give a clear and visual description of the order of tasks involved. Process design diagrams, which can include flowcharts, class diagrams and sequence diagrams among others, are frequently used to maximize the performance of the development process overall and to streamline the development process.

In the development of my application, I utilised two of these diagrams:

Sequence Diagram

Sequence diagrams are graphical depictions that show how various elements of a software system interact with one another. They are frequently used in software engineering to simulate how a system behaves over time and to spot any potential problems with the system's architecture. Messages transmitted and received between various objects or components, as well as their timing and order, are displayed in a sequence diagram. They are helpful for visualizing the interactions between many components of a system and can be used to depict both straightforward and complex scenarios. Sequence diagrams assist developers in comprehending the direction of control in a system and locating potential areas for system performance, reliability, and scalability enhancements.

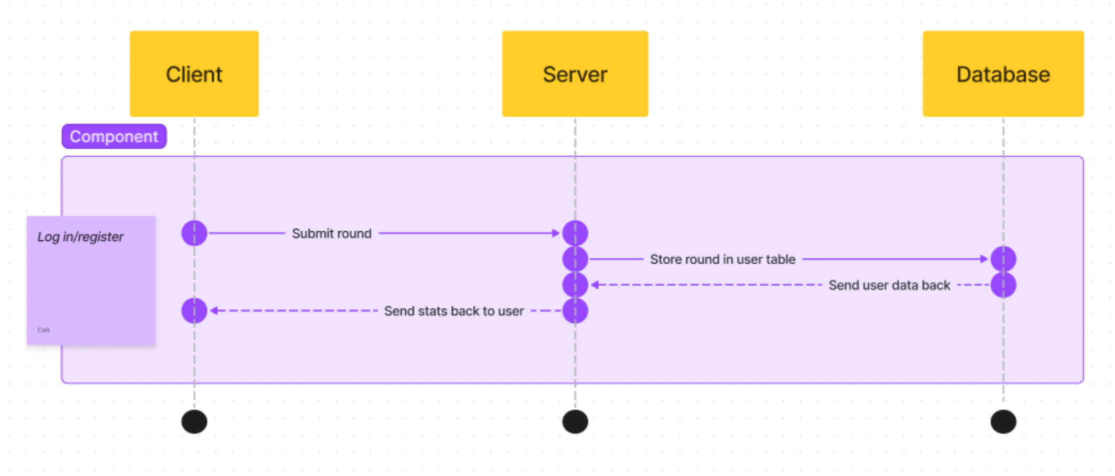


Figure 17 – Sequence Diagram

Shown above in Figure 17 is the Sequence Diagram I created to visualise the main functionality of my application.

Flow Chart

Flow charts are a great tool for representing a process's flow visually. They are diagrams that depict the steps of a procedure or algorithm, together with decision points, inputs, and outputs, using symbols and arrows. Flow charts enable developers to deconstruct complex processes into straightforward, understandable steps, which makes it easier to see possible problems or bottlenecks in the workflow. They are employed to model a variety of software systems, such as logic diagrams, process maps, and decision trees.

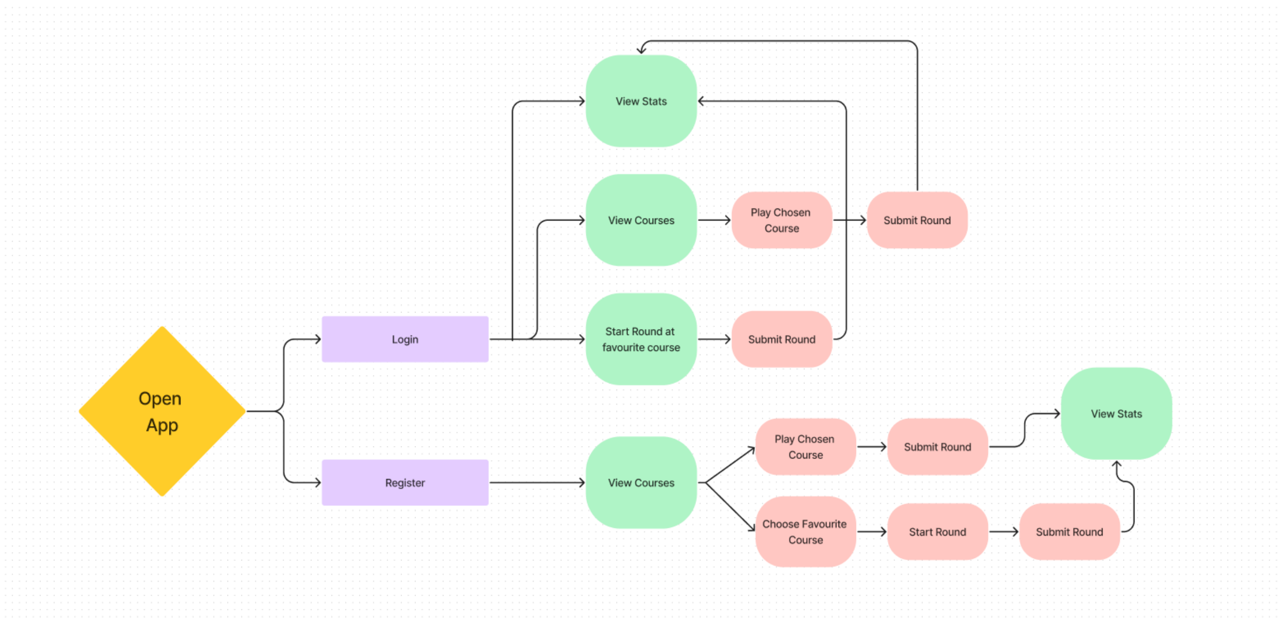


Figure 18 – Flow Chart for standard app usage

Shown above in Figure 18 is a flow chart outlining the expected general flow of the application being used.

4.3 User interface design

The basic goal of user interface design is to develop an interface that is intuitive, simple to use, and visually appealing. A good user interface design should allow users to execute activities quickly and easily, without feeling overwhelmed or confused. A well-designed user interface can improve the user experience, increase user engagement, and ultimately drive the software application's success.

4.3.1 Wireframe

Creating wireframes was an important part of the design process, it gave my application a visual baseline to work from writing the code. I made the wireframes for my project using Figma.

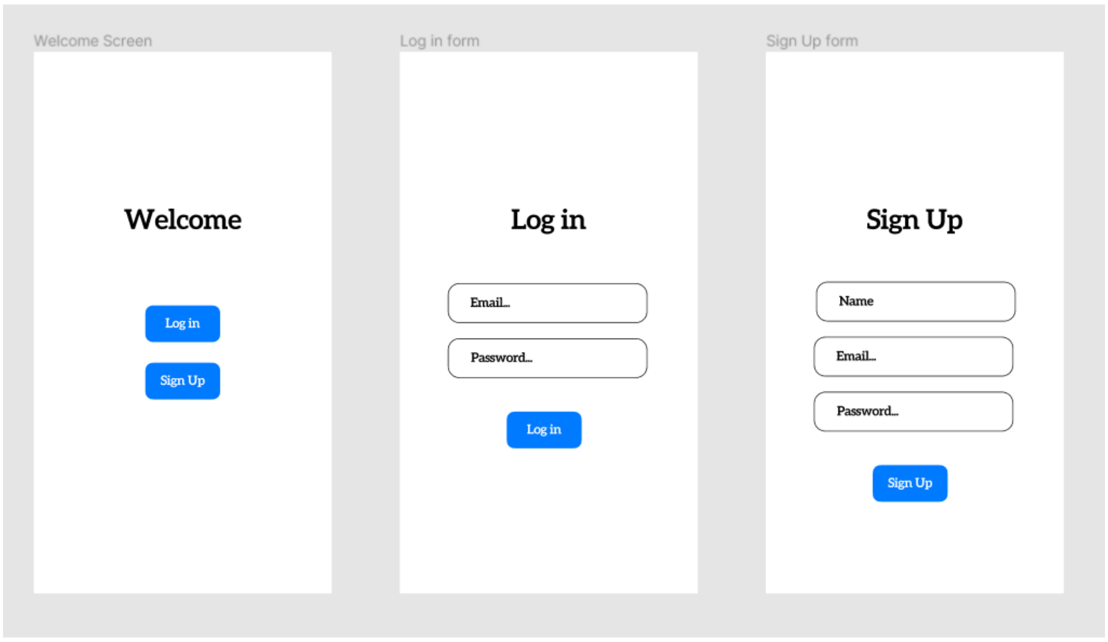


Figure 19 - Authentication Screen Wireframes

In Figure 19 above are the wireframes for the authentication screens on my application. When a user goes onto the app they are confronted with a welcome screen with log in and sign up options. They can then log in if they have an existing account or sign up. They will then be brought to the main app.

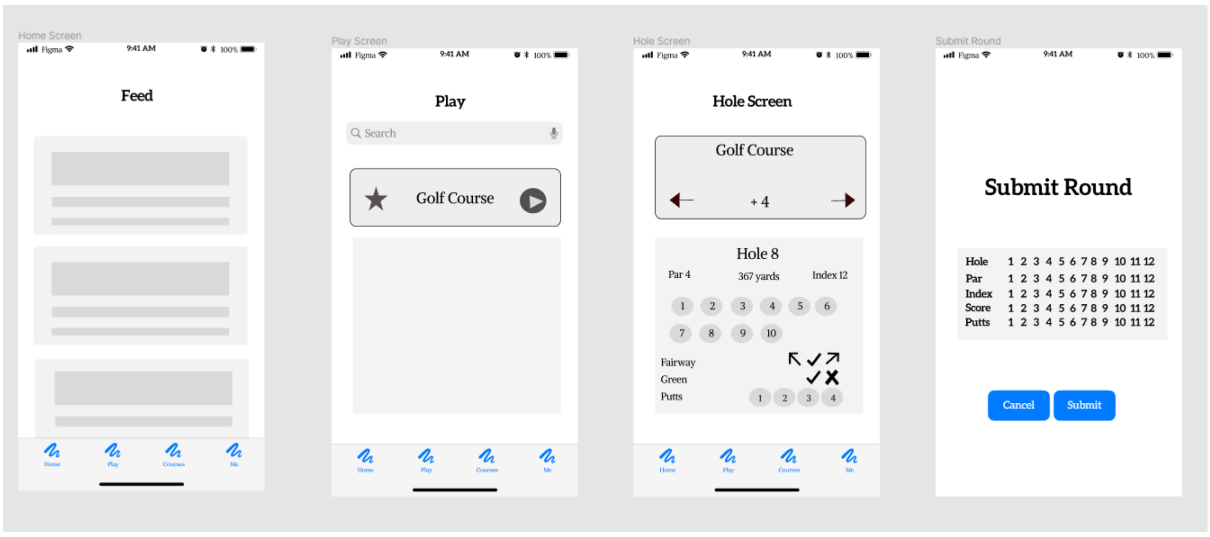


Figure 20 - Home, Play and Hole Screen Wireframes

Shown in Figure 20 above are wireframes for the Home Screen, Play Screen, Hole Screen and Submit Round Screen. The feed screen will display a list of rounds completed by the user. The Play Screen contains a search bar so users can search for their desired golf course. It will also display a user's favourite course for ease of access when they wish to play. Along with an interactive map of Golf Courses in Ireland. When a user chooses a course to play, they will be brought to the Hole Screen. This Screen takes in the chosen course's information and displays the details of every hole. Users can then enter their scores for each individual hole with their overall score being displayed and constantly updated at the top of the screen. When they have completed their 18 holes, they can then press submit round and they will be brought to the Submit Round Screen. This Screen contains a summary of the round they have played and the option to submit that round or return to the Hole Screen.

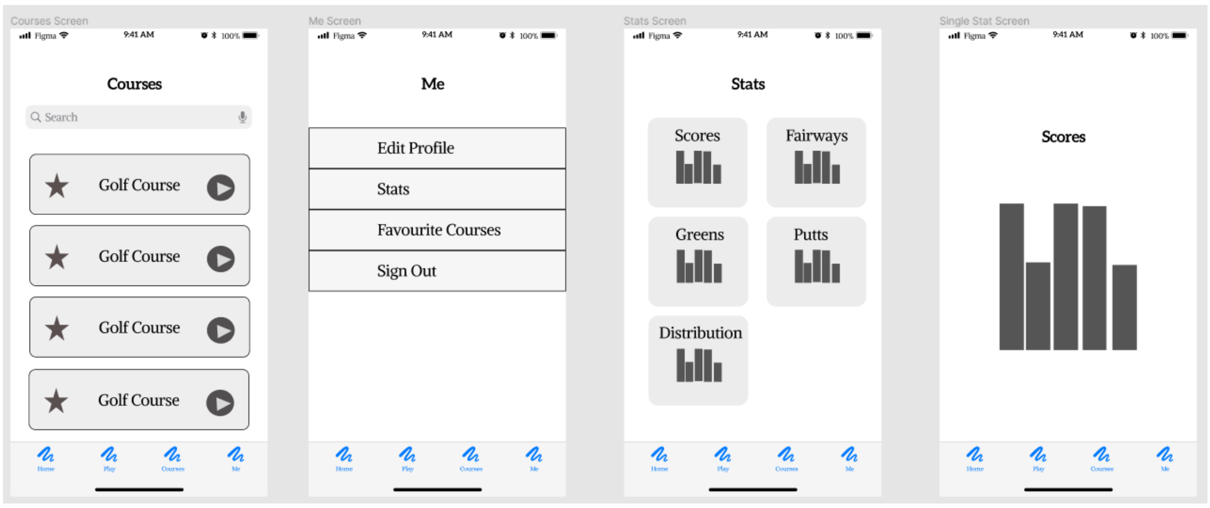


Figure 21 - Courses, Me and Stats Screen Wireframes

Shown in Figure 21 are the wireframes for the Courses Screen, Me Screen, Stats Screen and Individual Stat Screen. The Courses Screen displays a list of all the courses in the database, users can search for a specific course and see additional information about it. The Me Screen contains information about the user, they can edit their profile and view their stats. When they click the Stats button, they are brought to the Stats Screen, this contains a number of different metrics that they can view including their Scores over their past rounds, their percentage greens hit and the number of putts they took per hole. When they click on any of these options, they are brought to the Individual Stat Screen. This screen presents a graphical representation of the metric they have chosen to view.

4.3.2 Style guide

For the style of my application, I decided to use a scenic background for most screens I used standard font and utilised Material Community Icons to create an engaging graphical user interface while a user is filling out their scores.

4.4 Conclusion

In this chapter I outlined the processes and techniques I used to carry out both the Program Design and User Interface Design of my application. I talked about the technologies I decided to use and why they work well with each other. I discussed the structure of the folders in my project and created diagrams to gain a better understanding of all the necessary components I needed to make in my project and how they would interact with each other. This process gave my implementation a clearer plan and enabled me to have a clear purpose when coding and creating my database.

I created wireframes using Figma to make an initial UI design for my application and ensure I had a vision to work towards when implementing the front end of my application. I also decided on a style guide to work from to keep my app looking consistent throughout and visually appealing for users to keep them engaged.

5 Implementation

5.1 Introduction

This chapter describes the implementation for the application. The application has been developed using the following technologies:

- React Native

  I used React Native to develop the front end of my application. It is a JavaScript framework that is used for developing mobile applications and allows developers to write code once and deploy it across various platforms, such as iOS and Android. It is based on React, a JavaScript framework for creating user interfaces, and takes a similar approach to mobile app development.

- Express.js

  Express.js is a Node.js web application framework that simplifies the development of web apps and APIs. It includes capabilities like routing, middleware, template engines, and error handling that make it simple to build powerful and scalable web applications. Because Express.js is lightweight and adaptable, it is simple to combine with various third-party libraries and frameworks. It also has a big and active community that helps with its creation, upkeep, and support. Express.js is a popular framework for developing RESTful APIs, online apps, and server-side applications that require a simple and straightforward framework.

- MongoDB

  MongoDB is a NoSQL database made to manage unstructured data in a scalable and adaptable way. MongoDB uses a document-based data model as opposed to conventional relational databases, making it possible to store and retrieve complex data structures quickly and easily. It is a popular option for contemporary online and mobile apps because of its reputation for handling high volume and high velocity data with low latency.

  The application for this project is a Golf Score Tracking App, users can register log in and start a round of golf in their desired golf course. They can enter metrics for each individual hole such as their score, whether they hit the fairway, whether they hit the green in regulation and how many putts they took. When they are finished, they can submit their round and it is stored in the database. These metrics are saved, and users can view an array of different stats in the stats page. This gives user useful information on how they are performing on the course.

5.2 Scrum Methodology

The Scrum methodology revolves around a particular set of practices and roles incorporated in the development of software. A Scrum takes place in small periodic sections called Sprints, which are anywhere from one to four weeks long and conclude with feedback and reflection. All Sprints are their own separate entity that performs at least one tangible task that can be reviewed with ease.

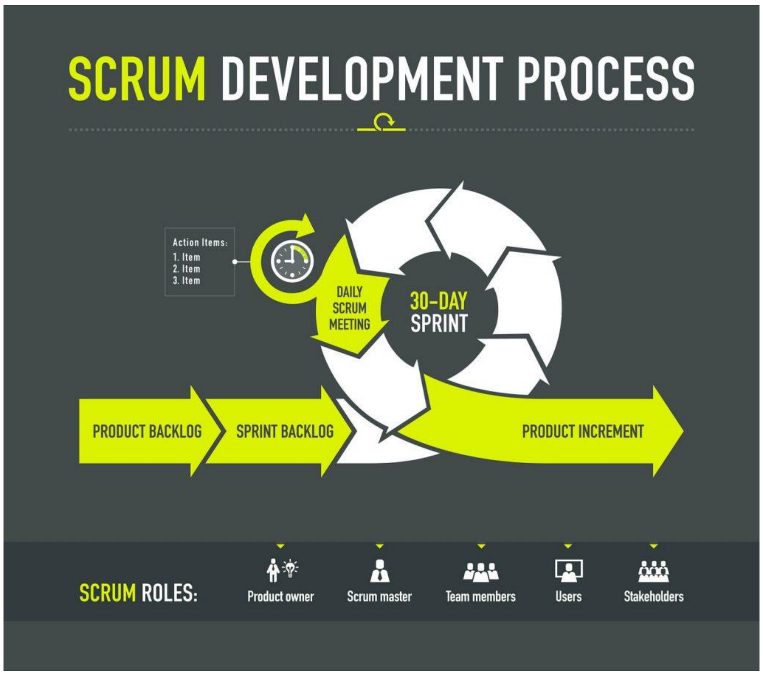


Figure 22 - Scrum development process

As Figure 22 above shows, the process is designed to complete a list of deliverables in a short period of time with evident accountability. The client is the one who prioritises objectives considering value and cost. The market requires quick, agile and flexible development of products to stay up to date with consumer needs and not be left in the dust of the rapidly developing tech industry. The scrum methodology is a great way to ensure these needs are met.

There are three main roles in a Scrum team:

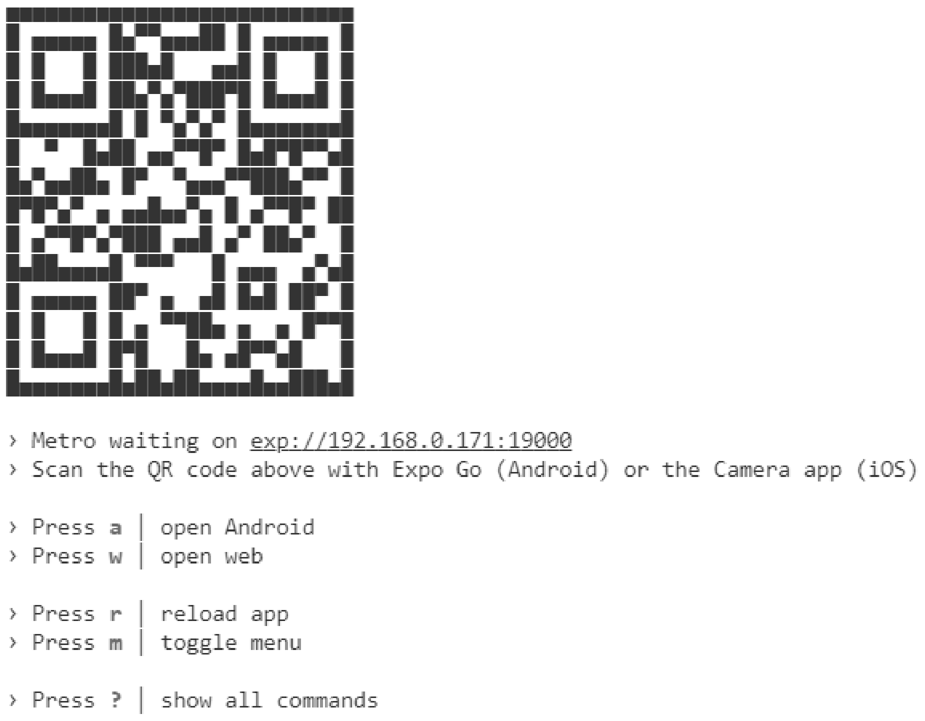
1. The Scrum master is the leader of the team and oversees keeping the scrum up to date, training, coaching and mentoring the team where needs be.
2. The Product owner is the representative of the stakeholders and customers that will use the software. Their main objective is maximising return on investment. They communicate the vision of the product to the team and prioritise the Product Backlog regularly.
3. The Team is the group of professionals with the required skills to translate the vision of the product to reality.

The implementation phase for this project consisted of eight sprints in total. Each sprint took place over a period of one week.

The requirements for the application were listed in a product backlog. Each item on the product backlog was broken down into a series of tasks which formed a sprint.

5.3 Development environment

I used Visual Studio Code to implement both the front end and the back end of my application. To run the React Native Expo app I downloaded the Expo Go app on my iPhone. I then ran 'npx expo start' in the node terminal on my PC. This built the code and ran a local server that I could connect to on my phone by scanning the QR code that it automatically creates.



```
› Metro waiting on exp://192.168.0.171:19000
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Press a │ open Android
› Press w │ open web

› Press r │ reload app
› Press m │ toggle menu

› Press ? │ show all commands
```

Figure 23 – Expo QR code in terminal

Scanning the code shown in Figure 23 above enabled me to run a real time version of my application and see updates as I edited the code. This was a fantastic way to develop a mobile app and I did not need to run an emulator from my PC. Which was ideal as my PC cannot handle that much.

5.4 Sprint 1

5.4.1 Goal

Describe which items on the product backlog form the tasks to be completed for this sprint.

The goal for Sprint 1 was to finish off my research document and begin to make some Paper Prototypes and Hi-Fi Prototypes. There was no coding during Sprint 1.

5.4.2 Item 1 – Completing Research Document

I finished writing my literature review that I had previously started in the previous term. I submitted this document and got approval from my supervisor that it was satisfactory so I could move on to the prototyping process for my project.

5.4.3 Item 2 – Paper Prototyping

We had a Paper Prototyping workshop in class during the week and we were handed out sheets of paper. I drew several rough drafts with pencil to get an initial idea of what some of my screens would look like. I chose a couple of my drafts that I thought looked the best and moved on to developing them further into a Hi-Fi Prototype

5.4.4 Item 3 – Hi-Fi Prototyping

I created a Hi-Fi Prototype using Figma. We had a Hi-Fi Prototyping workshop where I learned some useful techniques and tactics for developing these. I created all the components I needed for my prototype and was able to copy and use them wherever I needed and had the ability to easily edit and move things around. In the end I created what looked like a very sleek application without doing a single bit of coding.

5.5 Sprint 2

5.5.1 Goal

The goal of Sprint 2 was to create the initial version of my Design and Requirements documents. There was no coding done during Sprint 2.

5.5.2 Item 1 – Design Document

I began by thinking and deciding about how I wanted to structure my database and how the front end would interact with it. I developed an Application Architecture Block Diagram to visualise this interaction and clarify the individual components need to make up my project. I then began the process of creating an Entity Relationship Diagram, I decided upon which data tables I would need and what data I wanted to be stored in each. I thought about how I wanted all my entities to interact with each other. I then used Figma to draw this diagram. I followed the same type of process to create a Sequence Diagram and a User Flow Diagram.

5.5.3 Item 2 – Requirements Document

To develop my Requirements Document I first started by looking at applications that were similar to the one I wanted to create for this project. I then began gathering some input from prospective users of my application. I conducted some interviews and created a questionnaire for my friends that I knew had an interest in my project area. I then created some Proto Personas for a couple of different types of people that might use my app in order to get a sense of the userbase and their desires. After gathering this information, I made a list of functional and non-functional requirements for my application. When I had these requirements, I investigated the feasibility of creating the application with the technologies I had chosen.

5.6 Sprint 3

5.6.1 Goal

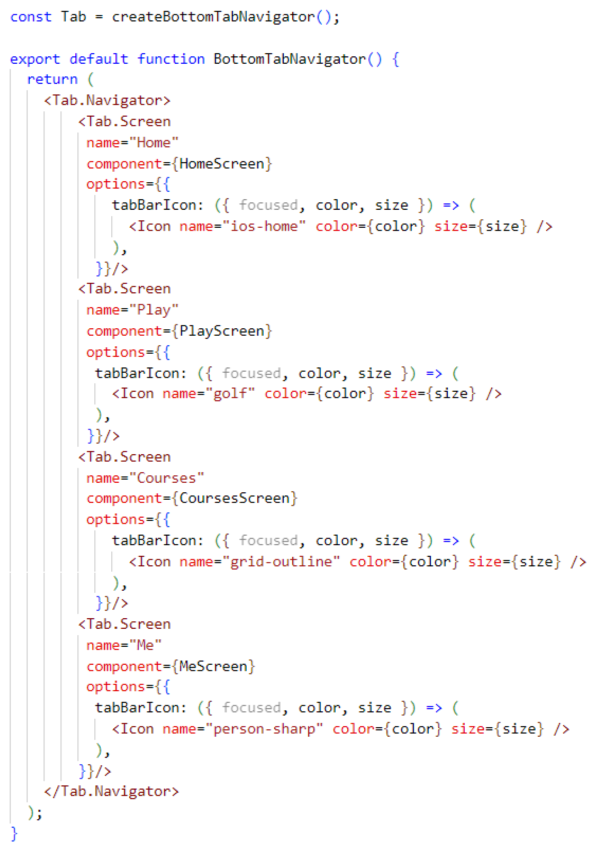
The goal of Sprint 3 was to begin implementation of my application.

5.6.2 Item 1 – Start working with React Native

While I had worked with React before, I had never worked with React Native. I started by creating a React Native Expo project and playing around with the functionality of the framework. Using the Expo app, I was able to get a development version of my project up on my iPhone. I was then able to easily work away on the code, quickly writing a simple "Hello World" function and have it show up on my mobile device.

5.6.3 Item 2 – Choose Libraries

I looked at different libraries for React Native development. I first looked at component libraries and decided I would use Native Base to create some of the components in my application. The first element of my application I wanted to create was the bottom tab navigator, this would allow me to move to the different screens in my app. I used the React Navigation library to create this and found it quite straightforward to implement.



```
const Tab = createBottomTabNavigator();

export default function BottomTabNavigator() {
  return (
    <Tab.Navigator>
      <Tab.Screen
        name="Home"
        component={HomeScreen}
        options={{
          tabBarIcon: ({ focused, color, size }) => (
            <Icon name="ios-home" color={color} size={size} />
          ),
        }}/>
      <Tab.Screen
        name="Play"
        component={PlayScreen}
        options={{
         tabBarIcon: ({ focused, color, size }) => (
          <Icon name="golf" color={color} size={size} />
         ),
        }}/>
      <Tab.Screen
        name="Courses"
        component={CoursesScreen}
        options={{
          tabBarIcon: ({ focused, color, size }) => (
            <Icon name="grid-outline" color={color} size={size} />
          ),
        }}/>
      <Tab.Screen
        name="Me"
        component={MeScreen}
        options={{
         tabBarIcon: ({ focused, color, size }) => (
          <Icon name="person-sharp" color={color} size={size} />
         ),
       }}/>
    </Tab.Navigator>
  );
}
```

Figure 24 - Bottom Tab navigator

Shown in Figure 24 above is the code for my Bottom Tab Navigator it was a simple adaptation from the React Navigation docs. I had set up the initial four screen components to use for the four main tabs I wanted to display and was able to nicely change the icons using Ionicons from React Native Vector Icons.

5.7 Sprint 4

5.7.1 Goal

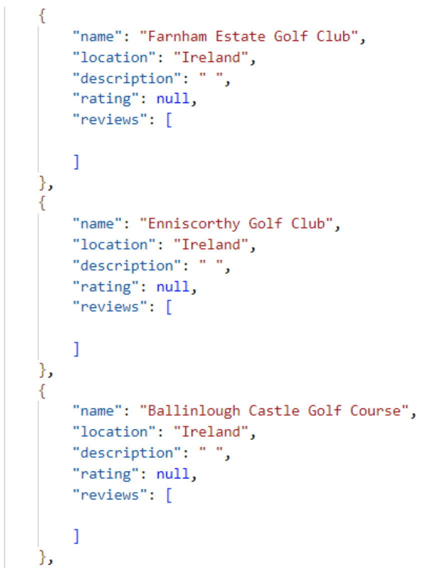
The goal of Sprint 4 was to update my Design Document and continue the initial implementation of my application and searching for a dataset to use.

5.7.2 Item 1 – Update Design Document

Having begun work on implementation I reviewed my Design Document to see if I was running into any issues. I realised I would need to add a few values to the user table to develop the application the way I wanted. I updated my design and was then able to continue implementation.

5.7.3 Item 2 – Fill out screens and creating data list.

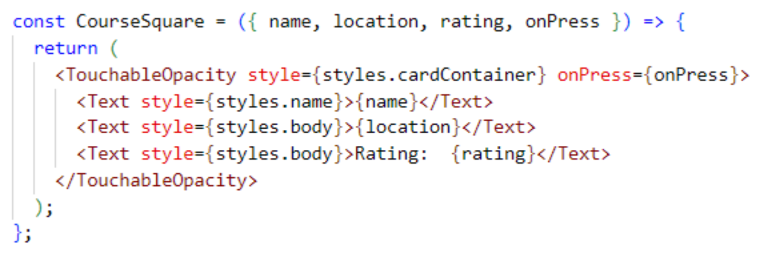
I continued work on the front end of my app by fleshing out the screens and adding components. I searched for an external API of golf courses in Ireland that I could read into my application. I was unable to find a suitable one for my needs and in order to keep the ball rolling I manually created a JSON document of 371 golf courses in Ireland.



```
{
    "name": "Farnham Estate Golf Club",
    "location": "Ireland",
    "description": " ",
    "rating": null,
    "reviews": [

    ]
},
{
    "name": "Enniscorthy Golf Club",
    "location": "Ireland",
    "description": " ",
    "rating": null,
    "reviews": [

    ]
},
{
    "name": "Ballinlough Castle Golf Course",
    "location": "Ireland",
    "description": " ",
    "rating": null,
    "reviews": [

    ]
},
```

Figure 25 - Section of golf courses dataset

In Figure 25 above is a small portion of the dataset I created initially. I fed this dataset straight into the front end in the assets folder initially, I then displayed them all in the courses screen using a list of card like components that I built.



```
const CourseSquare = ({ name, location, rating, onPress }) => {
  return (
    <TouchableOpacity style={styles.cardContainer} onPress={onPress}>
      <Text style={styles.name}>{name}</Text>
      <Text style={styles.body}>{location}</Text>
      <Text style={styles.body}>Rating:  {rating}</Text>
    </TouchableOpacity>
  );
};
```

Figure 26 - Card component for displaying a golf course

In Figure 26 above you can see how I created a simple card that reads in information from the courses screen.

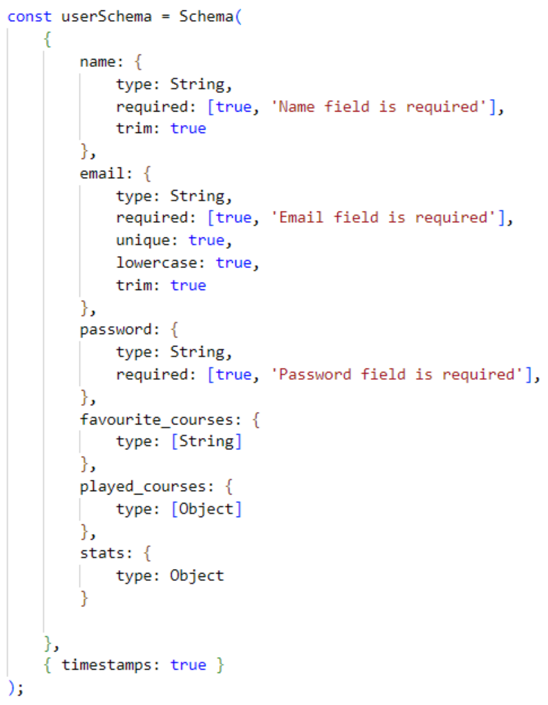
5.8 Sprint 5

5.8.1 Goal

The goal of Sprint 5 was to create my back end and database, connect it to the front end and implement authentication on the front end.

5.8.2 Item 1 – Create Back End and connect to Front End

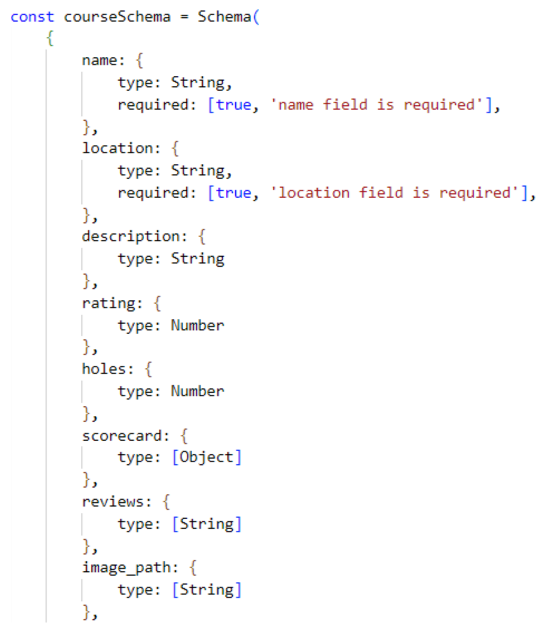
I used the template of a back end I had previously built for an earlier project to implement this step as quick as possible. I changed around the schemas for the courses and users to contain the values I required.



```
const userSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'Name field is required'],
            trim: true
        },
        email: {
            type: String,
            required: [true, 'Email field is required'],
            unique: true,
            lowercase: true,
            trim: true
        },
        password: {
            type: String,
            required: [true, 'Password field is required'],
        },
        favourite_courses: {
            type: [String]
        },
        played_courses: {
            type: [Object]
        },
        stats: {
            type: Object
        }

    },
    { timestamps: true }
);
```

Figure 27 - User Schema in Express

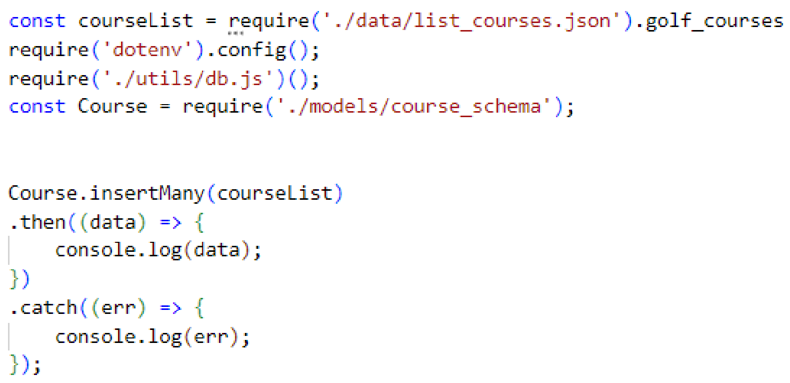
As you can see in Figure 27 above, I created six data values to be stored in the users table. Name, email and password along with a favourite courses and played courses array as well as a stats object to contain a user's overall metrics.



```
const courseSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'name field is required'],
        },
        location: {
            type: String,
            required: [true, 'location field is required'],
        },
        description: {
            type: String
        },
        rating: {
            type: Number
        },
        holes: {
            type: Number
        },
        scorecard: {
            type: [Object]
        },
        reviews: {
            type: [String]
        },
        image_path: {
            type: [String]
        },
```

Figure 28 - Courses Schema

Shown in Figure 28 above is the course schema, here I decided to have values for the name, location, description, rating and number of holes in the course. I also added arrays that held the scorecard of the course, reviews of the course and images. I decided not to implement reviews in the end because they were unnecessary.

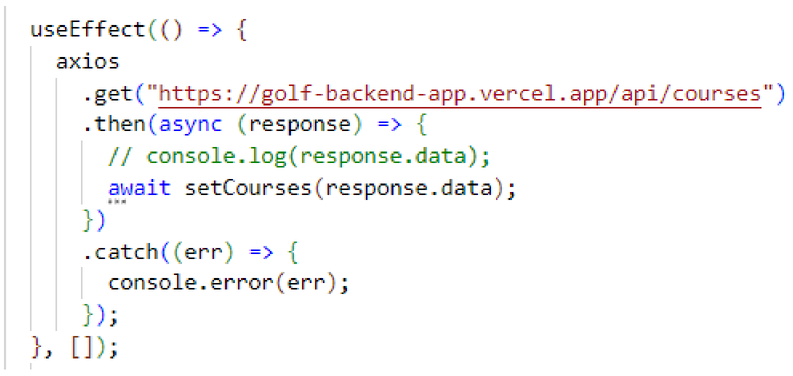


```
const courseList = require('./data/list_courses.json').golf_courses
require('dotenv').config();
require('./utils/db.js')();
const Course = require('./models/course_schema');


Course.insertMany(courseList)
.then((data) => {
    console.log(data);
})
.catch((err) => {
    console.log(err);
});
```

Figure 29 - seed.js file

Figure 29 above shows the seed.js file where I insert the courses into the MongoDB database. I then created a new MongoDB database and linked it to my back end. I seeded the database with the JSON list I created and then quickly hosted it using Vercel. I wanted to host it as quickly as possible so that I could immediately put the URLs into my REST client Insomnia and the Axios requests in the front end.

I changed the Courses Screen to send an Axios request to the back end in order to get the list of all the courses in my database.



```
useEffect(() => {
  axios
    .get("https://golf-backend-app.vercel.app/api/courses")
    .then(async (response) => {
      // console.log(response.data);
      await setCourses(response.data);
    })
    .catch((err) => {
      console.error(err);
    });
}, []);
```

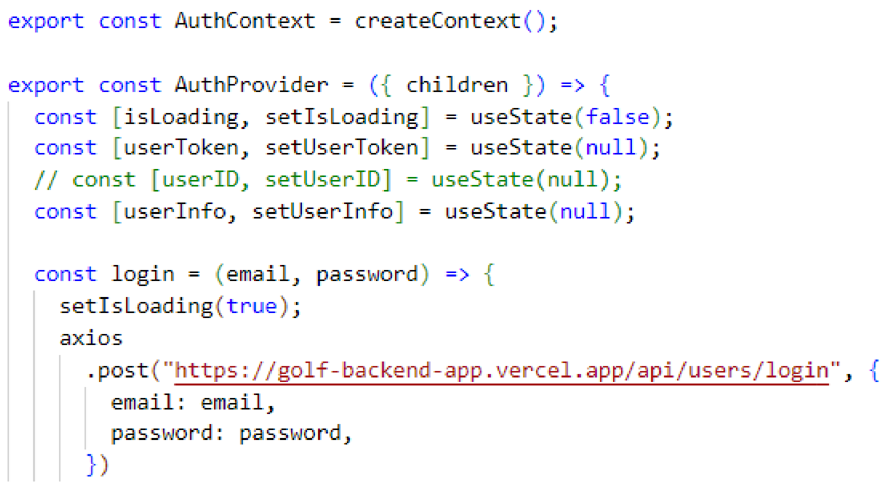
Figure 30 - Get courses Axios request to back end

Shown in Figure 30 above is the request to the hosted back end. This was a pain-free process and gave me a good base to work from to implement the rest of the application.

5.8.3 Item 2 – Authentication

I initially attempted to implement authentication the same way I did in previous React applications. I ran into problems implementing it this way, so I searched for a more appropriate way to do it. I found a very useful tutorial on YouTube that showed me an entirely new way to implement authentication. It involved creating a few new files and altering my App.js to incorporate them so that they control the application.

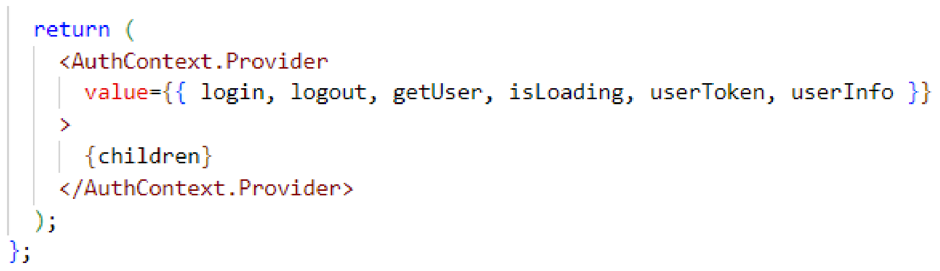
I created files called AppNav.js, AppStack.js, AuthStack.js and AuthContext.js.



```
export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isLoading, setIsLoading] = useState(false);
  const [userToken, setUserToken] = useState(null);
  // const [userID, setUserID] = useState(null);
  const [userInfo, setUserInfo] = useState(null);

  const login = (email, password) => {
    setIsLoading(true);
    axios
      .post("https://golf-backend-app.vercel.app/api/users/login", {
        email: email,
        password: password,
      })
```

Figure 31 - AuthContext.js code

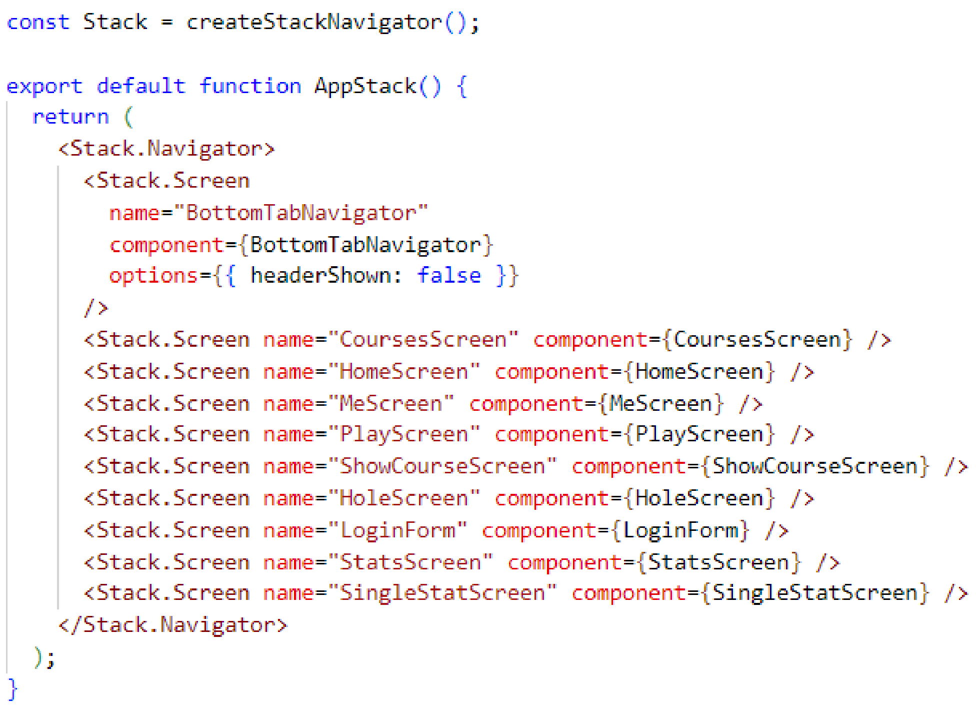
AuthContext.js shown above in Figure 31 contained all the of the authentication functionality inside the AuthProvider component, it does this using createContext which allowed data to be passed from this file to any other file in the program. It initialises a userToken value and contains the log in function for the app which sets the userToken value when a user logs in. This userToken value can be read from the AppNav.js file which then decides which stack to display to the user.



```
  return (
    <AuthContext.Provider
      value={{ login, logout, getUser, isLoading, userToken, userInfo }}
    >
      {children}
    </AuthContext.Provider>
  );
};
```

Figure 32 - AuthContext.js return value

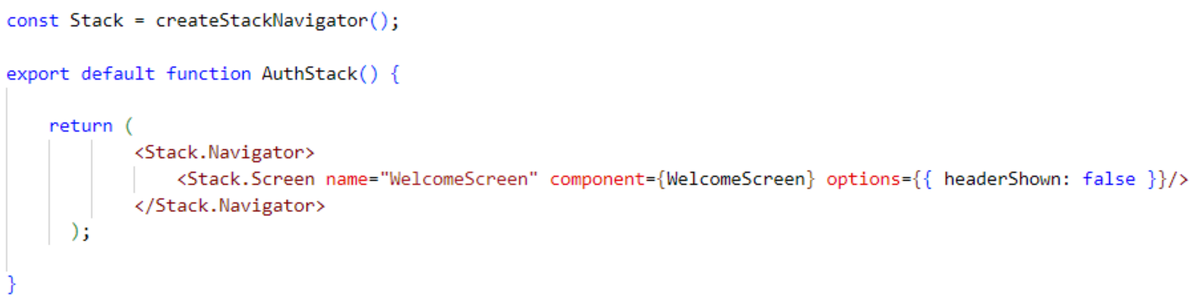
Shown in Figure 32 above is the return value for AuthContext. Here is where the values and functions are passed, enabling them to be read in from any other file in the program.



```
const Stack = createStackNavigator();

export default function AppStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="BottomTabNavigator"
        component={BottomTabNavigator}
        options={{ headerShown: false }}
      />
      <Stack.Screen name="CoursesScreen" component={CoursesScreen} />
      <Stack.Screen name="HomeScreen" component={HomeScreen} />
      <Stack.Screen name="MeScreen" component={MeScreen} />
      <Stack.Screen name="PlayScreen" component={PlayScreen} />
      <Stack.Screen name="ShowCourseScreen" component={ShowCourseScreen} />
      <Stack.Screen name="HoleScreen" component={HoleScreen} />
      <Stack.Screen name="LoginForm" component={LoginForm} />
      <Stack.Screen name="StatsScreen" component={StatsScreen} />
      <Stack.Screen name="SingleStatScreen" component={SingleStatScreen} />
    </Stack.Navigator>
  );
}
```

Figure 33 - AppStack file

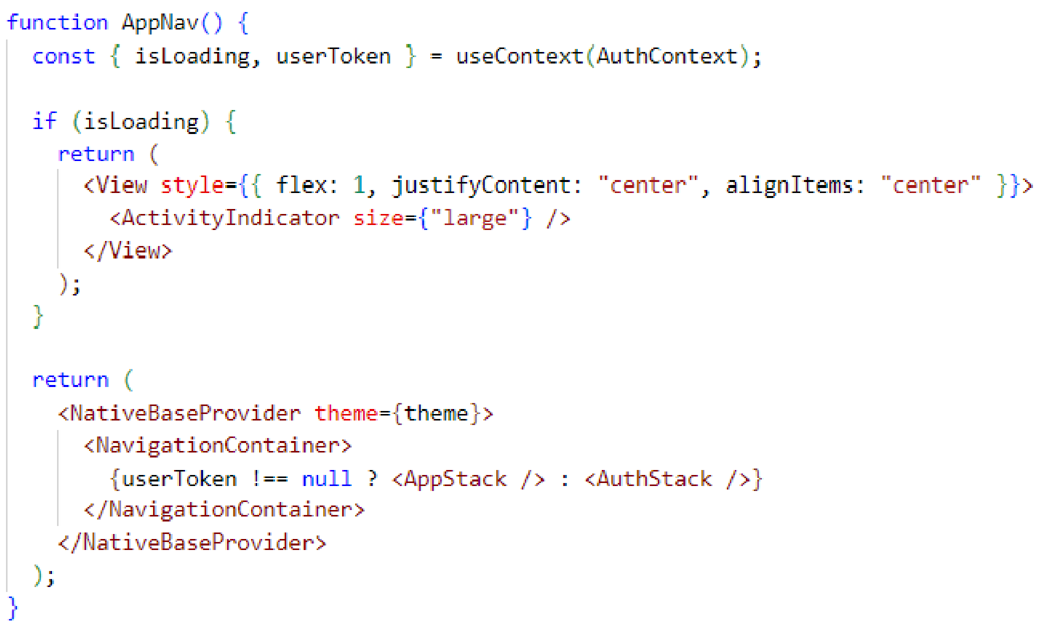
AppStack.js shown in Figure 33 above is comprised of the Stack Navigator containing all of the main screens in the application that I created to display when a user has been authenticated.



```
const Stack = createStackNavigator();

export default function AuthStack() {

    return (
            <Stack.Navigator>
                <Stack.Screen name="WelcomeScreen" component={WelcomeScreen} options={{ headerShown: false }}/>
            </Stack.Navigator>
        );

}
```

Figure 34 - AuthStack file

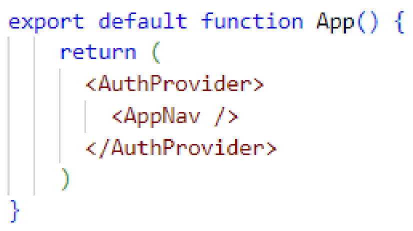
The AuthStack.js file shown in Figure 34 above is comprised of a Stack Navigator that only includes the Welcome Screen that I created, this is where a user will log in or register.



```
function AppNav() {
  const { isLoading, userToken } = useContext(AuthContext);

  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <ActivityIndicator size={"large"} />
      </View>
    );
  }

  return (
    <NativeBaseProvider theme={theme}>
      <NavigationContainer>
        {userToken !== null ? <AppStack /> : <AuthStack />}
      </NavigationContainer>
    </NativeBaseProvider>
  );
}
```

Figure 35 - AppNav file

AppNav.js shown in Figure 35 above is where I decide whether to show the AuthStack or the AppStack based on whether AuthContext provides a valid user token.

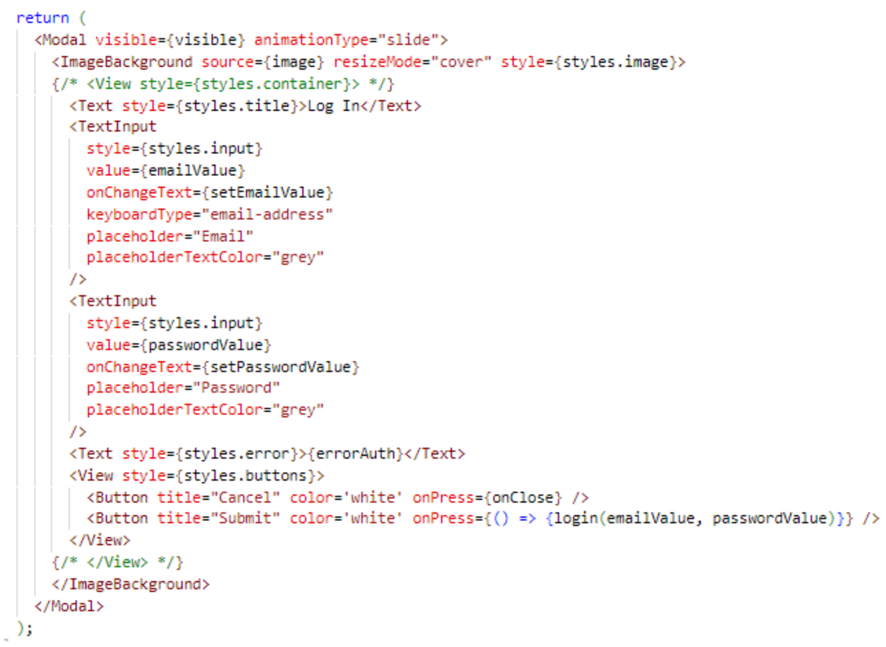


```
export default function App() {
    return (
      <AuthProvider>
        <AppNav />
      </AuthProvider>
    )
}
```

Figure 36 - App.js

The original App.js file that you can see in Figure 36 above, then just returns the AppNav component wrapped in the AuthProvider. App.js is then read in as normal as the main entry point to the application.

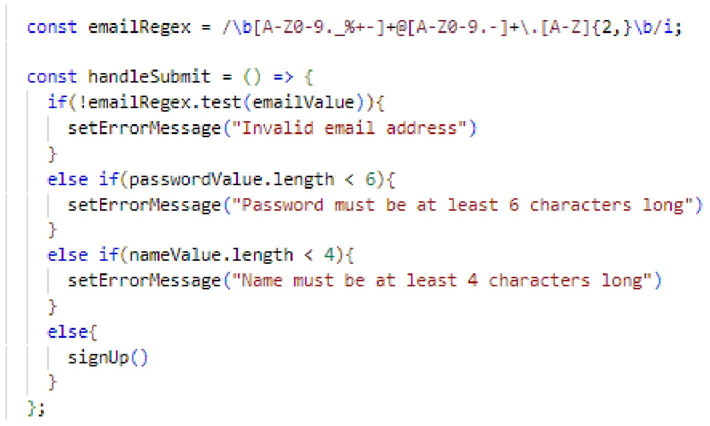
The next step was to create the log in and sign-up forms which would accessed in the Welcome Screen. I created these as modals which would become visible when a user pressed on either log in or sign up.



```
return (
  <Modal visible={visible} animationType="slide">
    <ImageBackground source={image} resizeMode="cover" style={styles.image}>
    {/* <View style={styles.container}> */}
      <Text style={styles.title}>Log In</Text>
      <TextInput
        style={styles.input}
        value={emailValue}
        onChangeText={setEmailValue}
        keyboardType="email-address"
        placeholder="Email"
        placeholderTextColor="grey"
      />
      <TextInput
        style={styles.input}
        value={passwordValue}
        onChangeText={setPasswordValue}
        placeholder="Password"
        placeholderTextColor="grey"
      />
      <Text style={styles.error}>{errorAuth}</Text>
      <View style={styles.buttons}>
        <Button title="Cancel" color='white' onPress={onClose} />
        <Button title="Submit" color='white' onPress={() => {login(emailValue, passwordValue)}} />
      </View>
    {/* </View> */}
    </ImageBackground>
  </Modal>
);
```

Figure 37 - Log in Modal

In Figure 37 above you can see the Modal code I created for the log in form. It incorporates two text input fields for email and password and two buttons for cancelling and submitting. The form retrieves the login function from the AuthContext file and calls the function when the submit button is pressed using the values filled out in the text fields. I also created an errorAuth value in AuthContext which will be displayed as error text if the user enters an invalid username or password. The sign up form is similar with their being an extra text input for the users name. It also provides more error feedback for form validation. Users name entered must be more than three characters long, their password must be more than five characters long and their email must be submitted in a proper email format. I used a regular expression test to check that emails were valid. This check is displayed in Figure 38 below.



```
const emailRegex = /\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b/i;

const handleSubmit = () => {
  if(!emailRegex.test(emailValue)){
    setErrorMessage("Invalid email address")
  }
  else if(passwordValue.length < 6){
    setErrorMessage("Password must be at least 6 characters long")
  }
  else if(nameValue.length < 4){
    setErrorMessage("Name must be at least 4 characters long")
  }
  else{
    signUp()
  }
};
```

Figure 38 - Regular Expression check for form

The function in Figure 38 above ensures a user signs up with proper information. When valid credentials are entered the register function is called to the server, the user is created and I then run the log in function to authorize them.

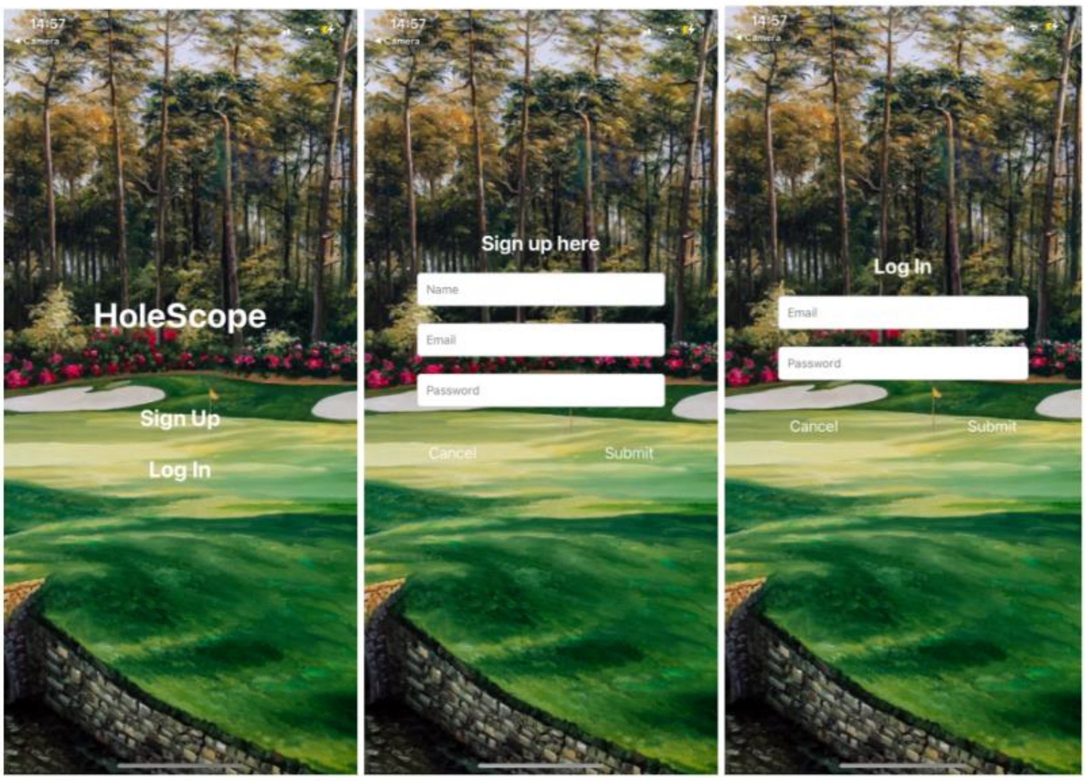


Figure 39 - Welcome Screen with Log in and sign up forms

Figure 39 shows the result of my Welcome Screen with its Log in and Sign up forms. For each Screen in my Project I used an ImageBackground component to lay all of my components on.

5.9 Sprint 6

5.9.1 Goal

The goal for Sprint 6 was to begin creating the playing functionality for the app.

5.9.2 Item 1 – Playing Functionality

I created the screen that encompassed the score entering aspect of the app. I called it the Hole Screen and it would read in the data from the chosen golf course. I initially created a Hole Card component that read in data from the scorecard array in the golf course's table. I designed this card to display information about the specific hole as well as the inputs for each value I wanted to track. The score on that hole, whether they hit the fairway, whether they hit the green and how many putts they took. I used icons for each value wrapping them in a Touchable Opacity component so that they acted like buttons.

I used Material Community Icons for this. I then realised that because there are three distinct types of holes in golf, i.e., par 3, par, 4 and par 5, the icons used for the score would have to look different for each type. I decided to then make three different types of hole card, I figured this way would be a bit cleaner than having a Hole Card file with lots of clunky conditional code in it. Figure 40 below isa look at what the cards look like.

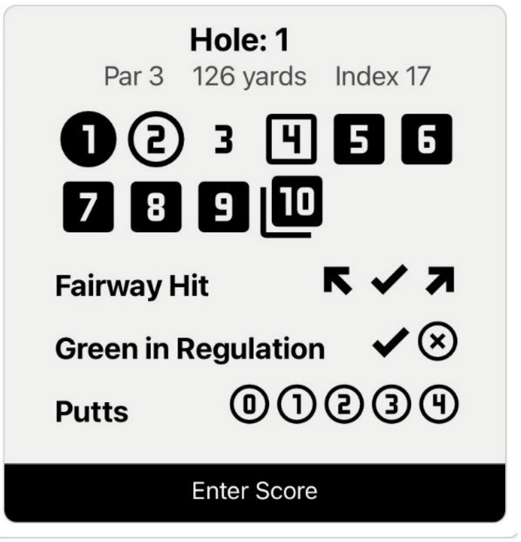
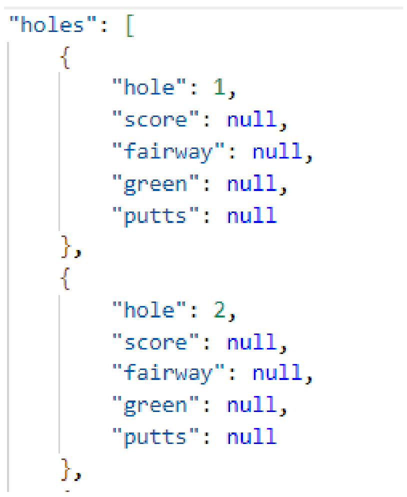


Figure 40 - Individual Hole Card

With the Hole Cards created I then went back to working on the hole screen. I read in the course data using route. params and was then able to access the scorecard array of the course. I then began to implement the functionality for moving from one hole to the next. I defined a currentHole value with useState and set it to zero initially which I used as the scorecard array index. I made two button components, Next and Previous which add or subtract one from the currentHole. I made them conditional so that a user cannot go lower than hole one or higher than hole eighteen as they don't exist, the buttons also go grey and unclickable when these limits are reached. I did this by conditionally returning the same button without the Touchable Opacity component and with grey styling. These buttons sit on top of the Hole Card.

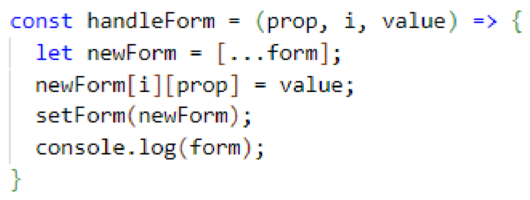
I then worked on reading the scores clicked on by users into a form. In the Hole Screen I created a form with useState and set it to an array of values set to null that I created.



```
"holes": [
    {
        "hole": 1,
        "score": null,
        "fairway": null,
        "green": null,
        "putts": null
    },
    {
        "hole": 2,
        "score": null,
        "fairway": null,
        "green": null,
        "putts": null
    },
```

Figure 41 - JSON of blank values

Figure 41 above shows a section of the JSON I created to act as a base for the form that the user fills out. I then made a handleForm function which takes in three parameters, “prop” the name of the type of input i.e. score, “i” the scorecard array index and “value” the value of the input.

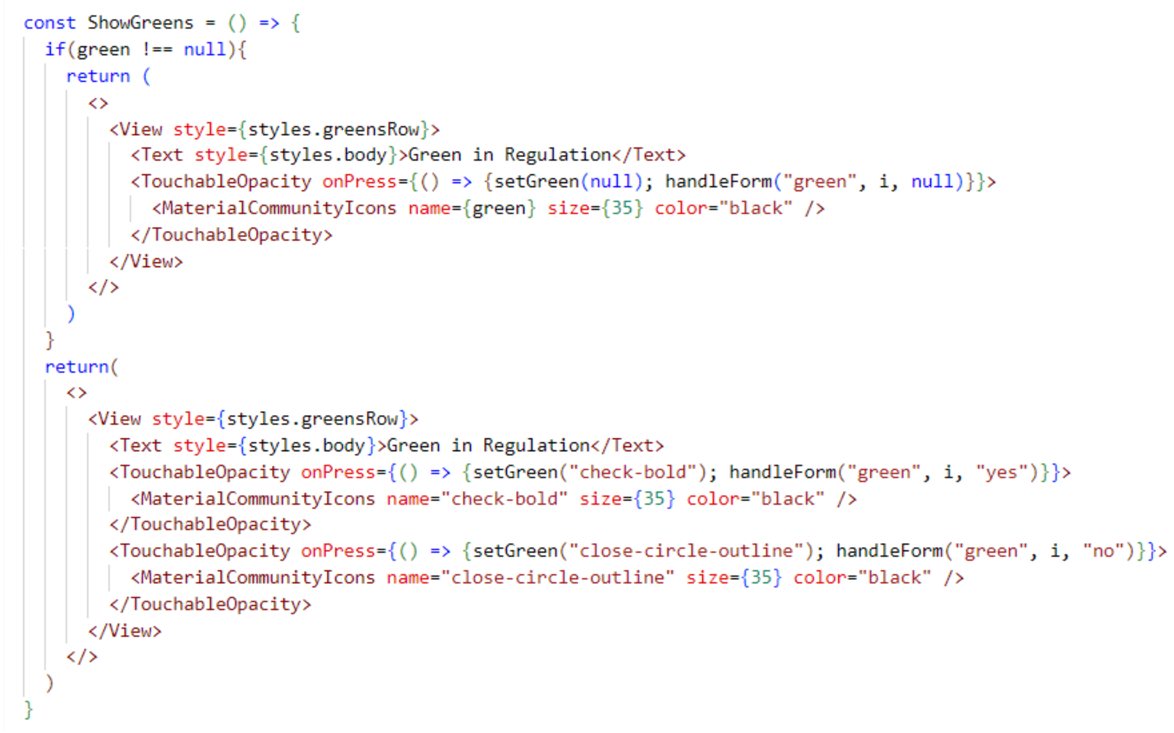


```
const handleForm = (prop, i, value) => {
  let newForm = [...form];
  newForm[i][prop] = value;
  setForm(newForm);
  console.log(form);
}
```

Figure 42 - handleForm function in hole screen

As you can see above in Figure 42 the handleForm function sets the form with the parameters passed into it.

I then passed this function into Hole Card components and set each icon button inside to call the function when they were pressed. To add the visual confirmation that a value had been chosen, I created components for each input type and made it so when an icon is pressed it becomes the only icon in that input section which is visible and become more prominent. To do this I created an initial value for each type with useState and called the set function whenever an icon was pressed.



```
const ShowGreens = () => {
  if(green !== null){
    return (
      <>
        <View style={styles.greensRow}>
          <Text style={styles.body}>Green in Regulation</Text>
          <TouchableOpacity onPress={() => {setGreen(null); handleForm("green", i, null)}}>
            <MaterialCommunityIcons name={green} size={35} color="black" />
          </TouchableOpacity>
        </View>
      </>
    )
  }
  return(
    <>
      <View style={styles.greensRow}>
        <Text style={styles.body}>Green in Regulation</Text>
        <TouchableOpacity onPress={() => {setGreen("check-bold"); handleForm("green", i, "yes")}}>
          <MaterialCommunityIcons name="check-bold" size={35} color="black" />
        </TouchableOpacity>
        <TouchableOpacity onPress={() => {setGreen("close-circle-outline"); handleForm("green", i, "no")}}>
          <MaterialCommunityIcons name="close-circle-outline" size={35} color="black" />
        </TouchableOpacity>
      </View>
    </>
  )
}
```

Figure 43 - ShowGreens component in the Hole Card

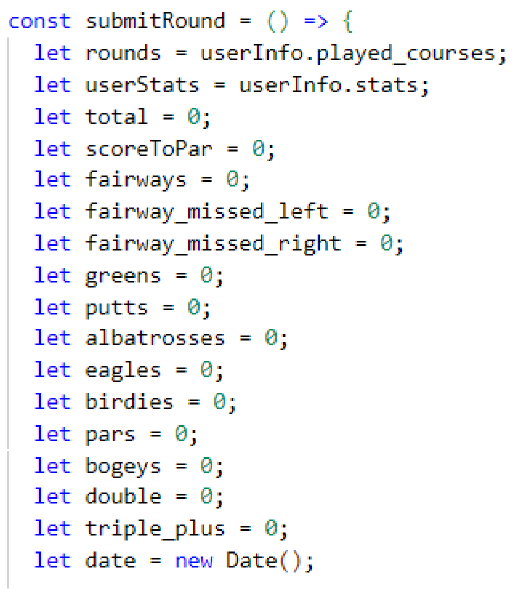
Shown above in Figure 43 is a look at the component I wrote for displaying the greens input. While the value is null it displays all the options and when an item is chosen it displays just that one. When an item is pressed again the value is set back to null so that a user can change their input.

I was then able to set the entire form in the Hole Screen and have an array full of data filled in by the user. The next step was to try submitting that data into the database.

5.9.3 Item 2 – Submitting Data

To submit the form entered by the user I started by getting the user's data from the database. I did this by calling the getUser function from the AuthContext file. When I had this information, I was then able to pull the played_courses array from the user. Every user when they sign up is given a stats object and a played_courses array which will contain every round they play. When a user submits a round, it will be pushed to their played_courses array and their stats from that round will be pushed to their stats object.

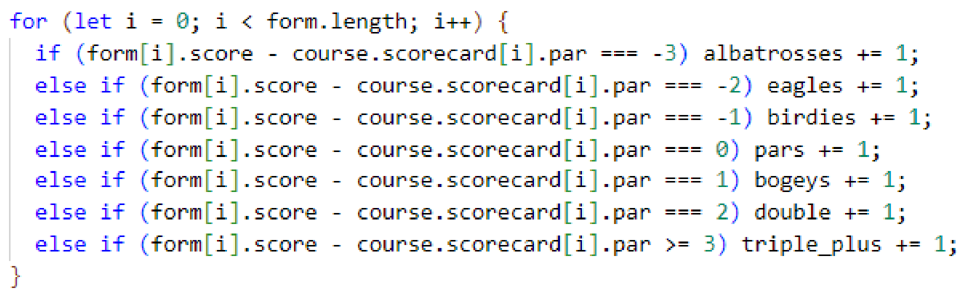
To do this I created a submitRound function and initialised several temporary variables. The first two were 'rounds' and 'stats' these were initialised with the user's played courses array and stats object, respectively. Then I made variables for each individual stat value e.g., total, fairways and putts. This can be seen in Figure 44 below.



```
const submitRound = () => {
  let rounds = userInfo.played_courses;
  let userStats = userInfo.stats;
  let total = 0;
  let scoreToPar = 0;
  let fairways = 0;
  let fairway_missed_left = 0;
  let fairway_missed_right = 0;
  let greens = 0;
  let putts = 0;
  let albatrosses = 0;
  let eagles = 0;
  let birdies = 0;
  let pars = 0;
  let bogeys = 0;
  let double = 0;
  let triple_plus = 0;
  let date = new Date();
```

Figure 44 - submitRound variable initialisation

I then went about calculating the values for each of these variables. I began by creating two different for loops. The first one calculated what the word value scores i.e. 'birdie', 'par' 'bogey' etc. were for each hole.

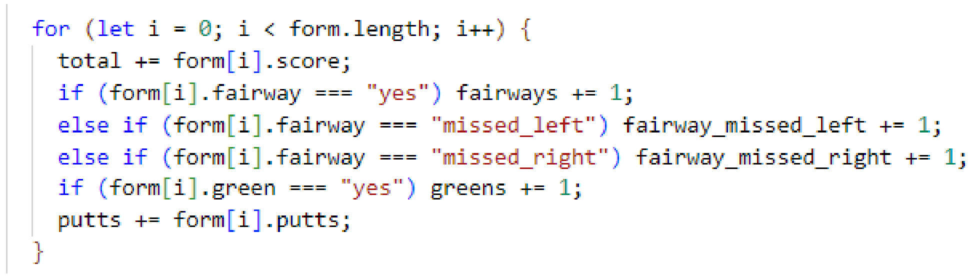


```
for (let i = 0; i < form.length; i++) {
  if (form[i].score - course.scorecard[i].par === -3) albatrosses += 1;
  else if (form[i].score - course.scorecard[i].par === -2) eagles += 1;
  else if (form[i].score - course.scorecard[i].par === -1) birdies += 1;
  else if (form[i].score - course.scorecard[i].par === 0) pars += 1;
  else if (form[i].score - course.scorecard[i].par === 1) bogeys += 1;
  else if (form[i].score - course.scorecard[i].par === 2) double += 1;
  else if (form[i].score - course.scorecard[i].par >= 3) triple_plus += 1;
}
```

Figure 45 - for loop calculating word values

In Figure 45 above you can see the loop I created. I see what number I get when I subtract the hole score filled out in the form by the par of that hole on the scorecard. I then decide which word value to give to it based on the number. So, a zero value will give a par, a minus one value will give a birdie and so on.

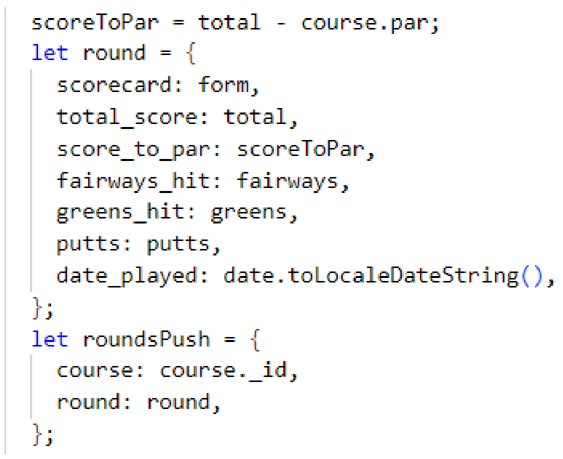
The second for loop I created calculated the values for total, fairways, greens and putts.



```
for (let i = 0; i < form.length; i++) {
  total += form[i].score;
  if (form[i].fairway === "yes") fairways += 1;
  else if (form[i].fairway === "missed_left") fairway_missed_left += 1;
  else if (form[i].fairway === "missed_right") fairway_missed_right += 1;
  if (form[i].green === "yes") greens += 1;
  putts += form[i].putts;
}
```

Figure 46 - for loop calculating more values

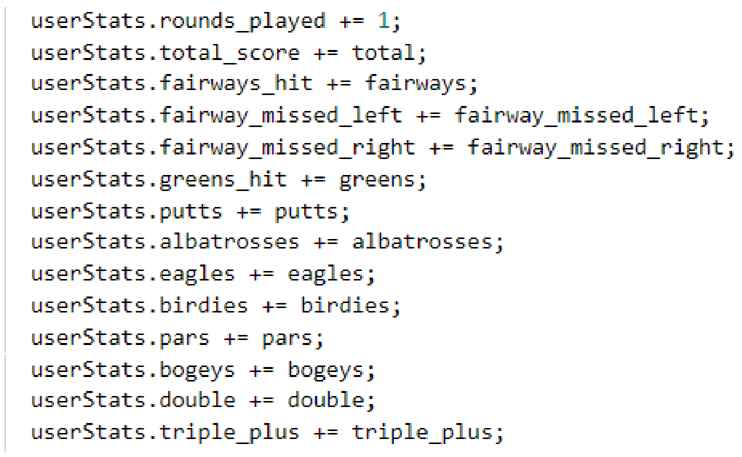
The for loop shown in Figure 46 above calculates these with some simple operations, first adding the form score for each hole onto the total score variable. Then I do a small series of if statements to determine the fairway values and an if statement to check and add onto the greens value. Finally, I add the putts on each form value to the overall putts variable. I then prepared all these values for entry into the database.



```
scoreToPar = total - course.par;
let round = {
  scorecard: form,
  total_score: total,
  score_to_par: scoreToPar,
  fairways_hit: fairways,
  greens_hit: greens,
  putts: putts,
  date_played: date.toLocaleDateString(),
};
let roundsPush = {
  course: course._id,
  round: round,
};
```

Figure 47 - Entry prep for submitting round

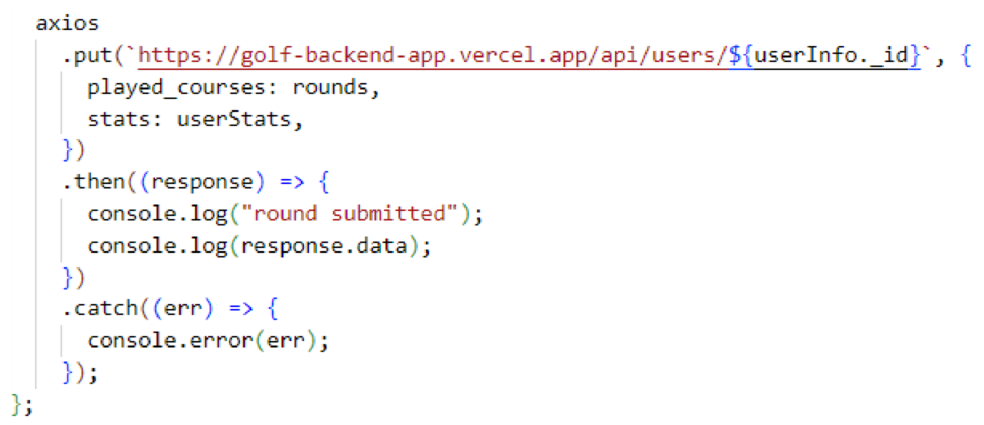
As you can see in Figure 47 above, I created another temporary variable called round which I set as an object containing the form and the stat values to go along with it. I then made another object called roundsPush which contained the course id of the course being played and the round object I just created before. Then before submitting I push this roundsPush object to the rounds array I initialised at the start of the function. I then added the overall stats to the userStats object.



```
userStats.rounds_played += 1;
userStats.total_score += total;
userStats.fairways_hit += fairways;
userStats.fairway_missed_left += fairway_missed_left;
userStats.fairway_missed_right += fairway_missed_right;
userStats.greens_hit += greens;
userStats.putts += putts;
userStats.albatrosses += albatrosses;
userStats.eagles += eagles;
userStats.birdies += birdies;
userStats.pars += pars;
userStats.bogeys += bogeys;
userStats.double += double;
userStats.triple_plus += triple_plus;
```

Figure 48 - Adding to userStats object

Shown in Figure 48 above is how I added all of the individual stats to the userStats object in preparation for pushing it to the database. The next step was making the API call.

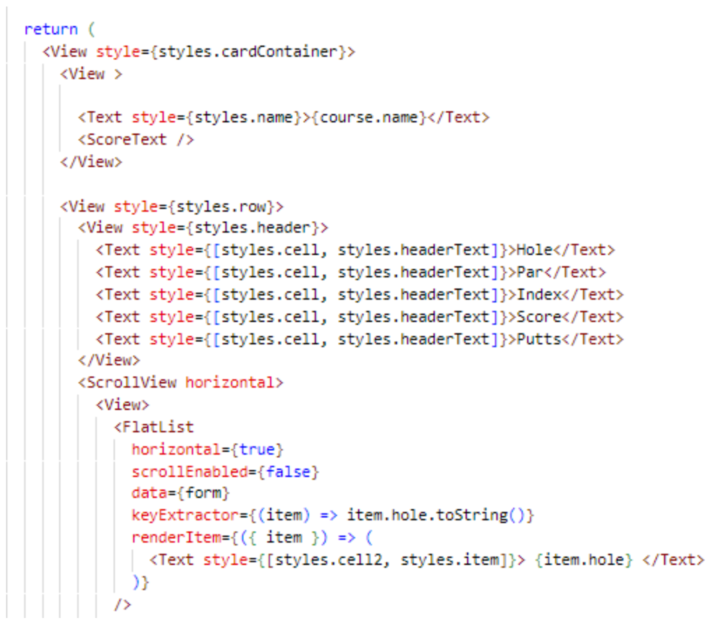


```
  axios
    .put(`https://golf-backend-app.vercel.app/api/users/${userInfo._id}`, {
      played_courses: rounds,
      stats: userStats,
    })
    .then((response) => {
      console.log("round submitted");
      console.log(response.data);
    })
    .catch((err) => {
      console.error(err);
    });
};
```

Figure 49 - API call to push round and stats

I make an axios put request to my back end shown above in Figure 49. I simply set the played_courses and stats values to the ones I prepared throughout the function to that point.

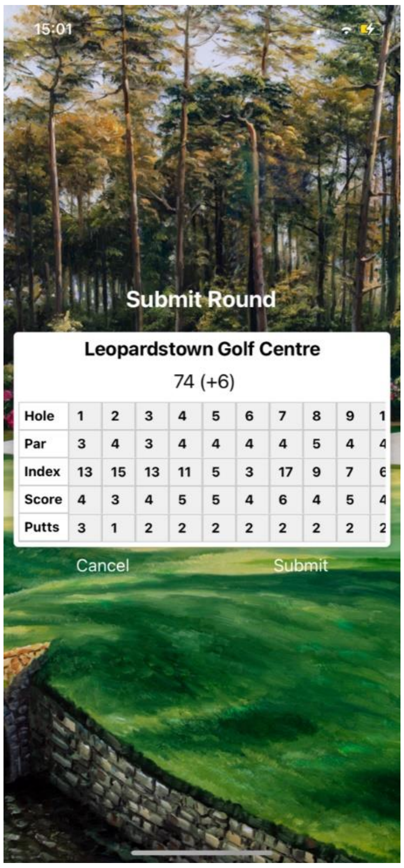
The next step was to create the submit round form that users would be brought to when they finished their round. This form would display the round they had just filled out in a table and the option to submit or cancel. I created a modal similar to the log in and sign-up forms. I created a RoundReview component that takes in the form filled out by the user and the ID of the course they had played. To create the table, I made a row and then created five headings that would be displayed in a column, I then created a ScrollView that contained five FlatLists which each contained the relevant hole by hole information for each heading. This made a table like component that allowed a user to scroll horizontally and view each corresponding stat. A portion of this code can be seen below in Figure 50.



```
return (
  <View style={styles.cardContainer}>
    <View >

      <Text style={styles.name}>{course.name}</Text>
      <ScoreText />
    </View>

    <View style={styles.row}>
      <View style={styles.header}>
        <Text style={[styles.cell, styles.headerText]}>Hole</Text>
        <Text style={[styles.cell, styles.headerText]}>Par</Text>
        <Text style={[styles.cell, styles.headerText]}>Index</Text>
        <Text style={[styles.cell, styles.headerText]}>Score</Text>
        <Text style={[styles.cell, styles.headerText]}>Putts</Text>
      </View>
      <ScrollView horizontal>
        <View>
          <FlatList
            horizontal={true}
            scrollEnabled={false}
            data={form}
            keyExtractor={(item) => item.hole.toString()}
            renderItem={({ item }) => (
              <Text style={[styles.cell2, styles.item]}> {item.hole} </Text>
            )}
          />
```

Figure 50 - Code for RoundReview component

The RoundReview component was then able to be placed in the Submit Round Form. The form's submit button then calls the submit round function I created in the Hole Screen and the user's round is submitted to the database. The result of this form can be seen below in Figure 51.



Leopardstown Golf Centre

74 (+6)

Hole	1	2	3	4	5	6	7	8	9	1
Par	3	4	3	4	4	4	4	5	4	4
Index	13	15	13	11	5	3	17	9	7	6
Score	4	3	4	5	5	4	6	4	5	4
Putts	3	1	2	2	2	2	2	2	2	2

Figure 51 - SubmitRound Form with RoundReview Component

Now that users could submit their rounds to the database the next step was to begin displaying their stats.

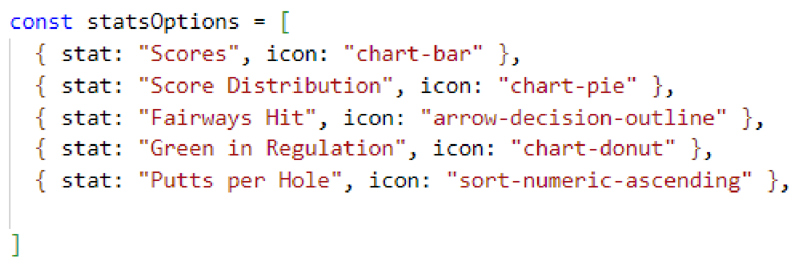
5.10 Sprint 7

5.10.1 Goal

The goal of Sprint 7 was to begin displaying the users' stats.

5.10.2 Item 1 – Create Stats Screen

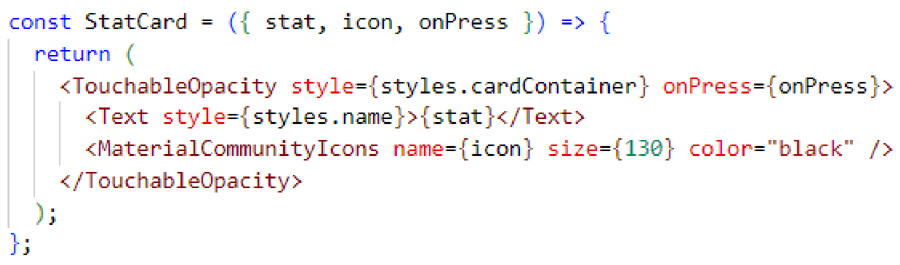
I wanted the Stats Screen to include a list of cards that represented each stat where users can click on the stat they wish to view. To begin process I had a look for icons that I thought would represent each stat well, I used icons from the Material Community Icons library. I then created an array of objects, each object represented an individual stat with a value representing the stat's name and the name of the icon that went along with it.



```
const statsOptions = [
  { stat: "Scores", icon: "chart-bar" },
  { stat: "Score Distribution", icon: "chart-pie" },
  { stat: "Fairways Hit", icon: "arrow-decision-outline" },
  { stat: "Green in Regulation", icon: "chart-donut" },
  { stat: "Putts per Hole", icon: "sort-numeric-ascending" },

]
```

Figure 52 - statsOptions Array

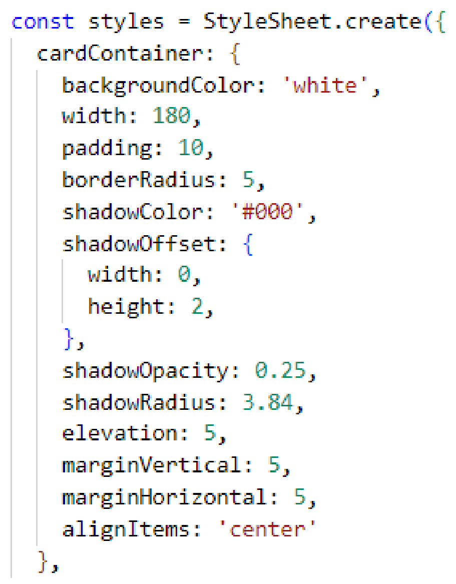
This array can be seen above in Figure 52. Then I created a StatsCard component which I would use to display these stats individually.



```
const StatCard = ({ stat, icon, onPress }) => {
  return (
    <TouchableOpacity style={styles.cardContainer} onPress={onPress}>
      <Text style={styles.name}>{stat}</Text>
      <MaterialCommunityIcons name={icon} size={130} color="black" />
    </TouchableOpacity>
  );
};
```

Figure 53 - StatsCard Component

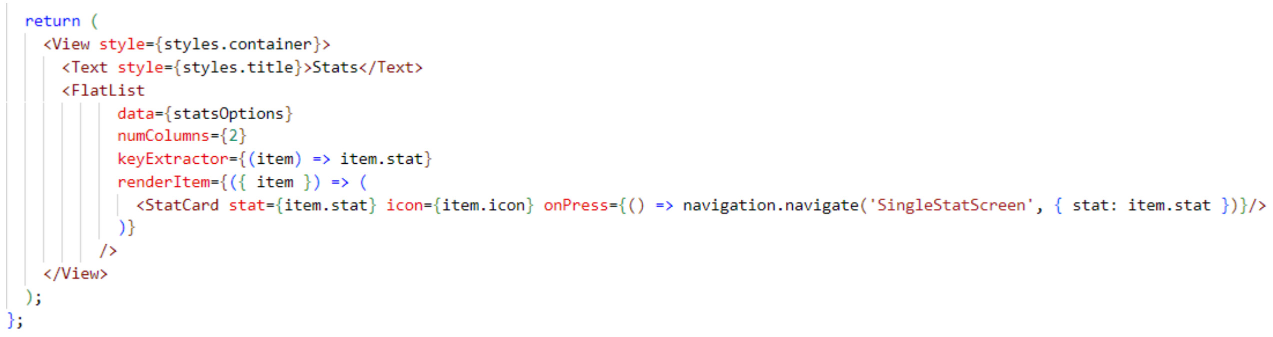
Figure 53 above shows the StatsCard component I created. The component takes in two values and a function, the name of the stat, its associated icon name and the onPress function used to bring the user to the Individual Stats Screen. The component is made up of a simple TouchableOpacity component with text representing the stat's name and its icon. I used styling to create a card looking element. This styling can be seen below in Figure 54.



```
const styles = StyleSheet.create({
  cardContainer: {
    backgroundColor: 'white',
    width: 180,
    padding: 10,
    borderRadius: 5,
    shadowColor: '#000',
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5,
    marginVertical: 5,
    marginHorizontal: 5,
    alignItems: 'center'
  },
```

Figure 54 - Card Styling

Now that I had a reusable and adaptable card component, I could display each stat as a card on the Stats Screen using a FlatList.



```
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Stats</Text>
      <FlatList
          data={statsOptions}
          numColumns={2}
          keyExtractor={(item) => item.stat}
          renderItem={({ item }) => (
            <StatCard stat={item.stat} icon={item.icon} onPress={() => navigation.navigate('SingleStatScreen', { stat: item.stat })}/>
          )}
        />
    </View>
  );
};
```

Figure 55 - Stats Screen return value

As you can see in Figure 55 above, I used a FlatList to create a 2-column grid displaying the stat cards. I used the statsOptions array I initialised and fed the values from each item into a StatCard component. The result can be seen in Figure 56 below.

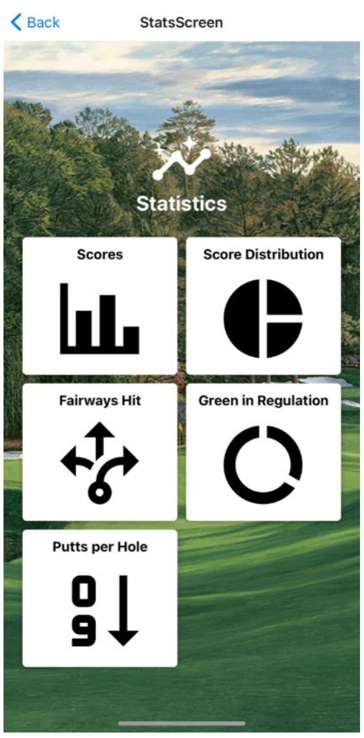
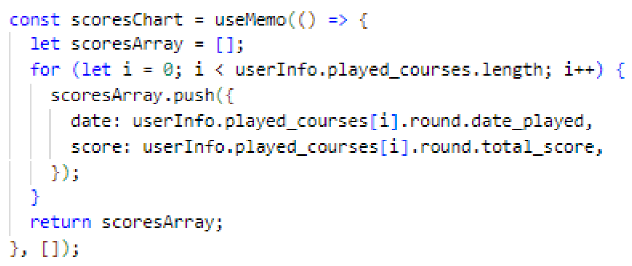


Figure 56 - Stats Screen

5.10.3 Item 2 – Show Stats

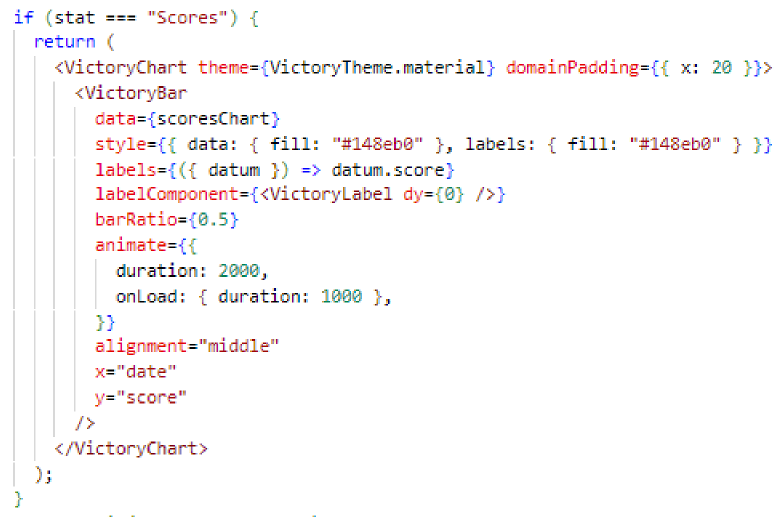
Now that I had the main Stats Screen working, I could get to work on making the Screen for the individual stats. I looked through a couple of options for chart libraries and I decided upon Victory to create the graphics for my stats. I then made a SingleStat Screen to display one stat at a time. To make the code for this screen nice and clean I also created a ShowStat component that would contain most of the functionality for charting the stats. The first stat I worked on charting was the Scores stat, this would be a bar chart of overall scores from a user's rounds with the score on the y axis and the date they played the round on the x axis. To do this I had to create an array from the user's data that would be suitable to display in a VictoryBar component. In order to create a computed array, I used the useMemo hook to initialise a scoresChart array.



```
const scoresChart = useMemo(() => {
  let scoresArray = [];
  for (let i = 0; i < userInfo.played_courses.length; i++) {
    scoresArray.push({
      date: userInfo.played_courses[i].round.date_played,
      score: userInfo.played_courses[i].round.total_score,
    });
  }
  return scoresArray;
}, []);
```

Figure 57 - scoresChart array initialisation

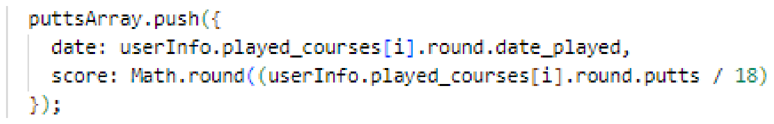
Shown in Figure 57 above is the scoresChart initialisation. I created a temporary array variable and used a for loop to go through the users played_courses array to push each round's date and score to the new array. I then return the new array and was able to plug it in to the VictoryBar.



```
if (stat === "Scores") {
  return (
    <VictoryChart theme={VictoryTheme.material} domainPadding={{ x: 20 }}>
      <VictoryBar
        data={scoresChart}
        style={{ data: { fill: "#148eb0" }, labels: { fill: "#148eb0" } }}
        labels={({ datum }) => datum.score}
        labelComponent={<VictoryLabel dy={0} />}
        barRatio={0.5}
        animate={{
          duration: 2000,
          onLoad: { duration: 1000 },
        }}
        alignment="middle"
        x="date"
        y="score"
      />
    </VictoryChart>
  );
}
```

Figure 58 - Scores Bar Chart

Figure 58 above shows the VictoryBar component wrapped in a VictoryChart component. I return this chart if and when the stat passed into the ShowStat component is equal to “Scores”. I used an almost identical method to create the Putts Chart the only difference is that instead of displaying the overall number of putts a user took in the round, I wanted to display the average number of putts they took per hole. I thought this was a more useful metric. To do this I used a Math.round function when pushing to the putts array.



```
puttsArray.push({
  date: userInfo.played_courses[i].round.date_played,
  score: Math.round((userInfo.played_courses[i].round.putts / 18)
});
```

Figure 59 - puttsArray push function

As you can see in Figure 59 above, I simply divide the total number of putts by 18 (holes) and put that in a Math.round function to limit the number to one decimal place. The result of these two charts can be seen below in Figure 60.

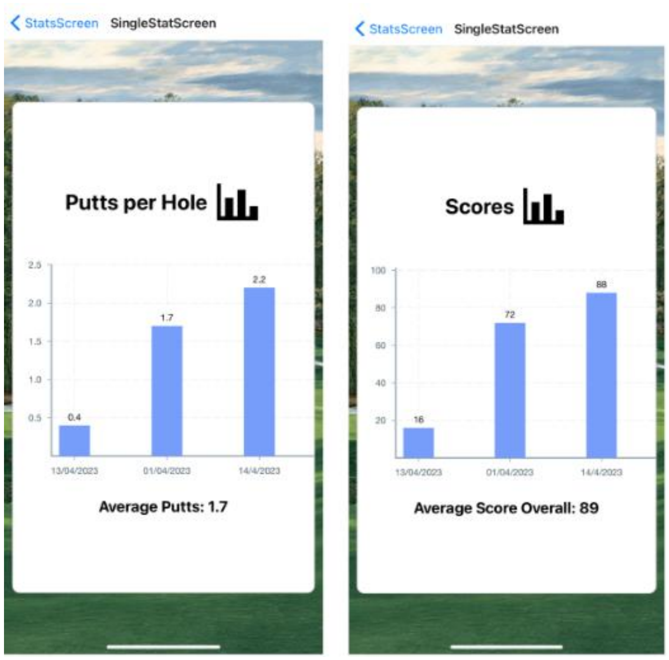


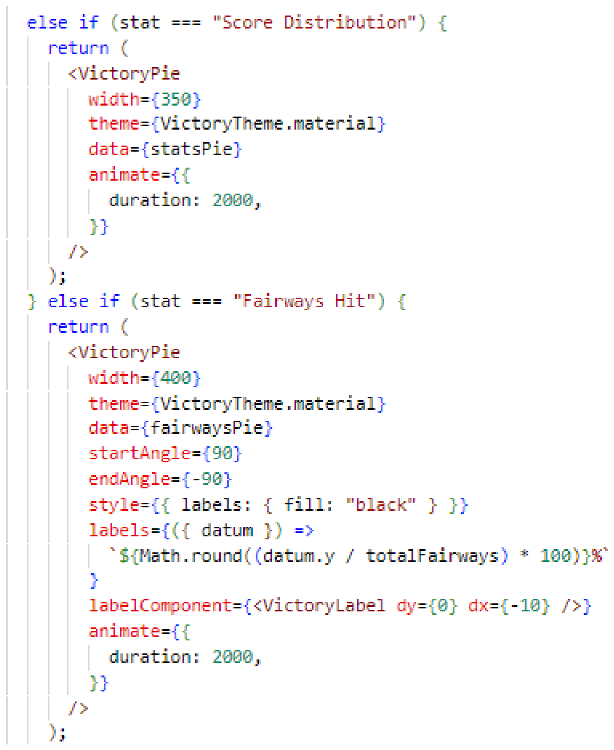
Figure 60 - Scores and Putts Charts

I then created pie charts using VictoryPie components, for the Score Distribution and Fairways Hit.

```
const statsPie = [
  { x: "Eagle +", y: userInfo.stats.eagles + userInfo.stats.albatrosses },
  { x: "Bogey", y: userInfo.stats.bogeys },
  { x: "Birdie", y: userInfo.stats.birdies },
  { x: "Double +", y: userInfo.stats.double + userInfo.stats.triple_plus },
  { x: "Par", y: userInfo.stats.pars },
];

const fairwaysPie = [
  { x: "Right", y: userInfo.stats.fairway_missed_right },
  { x: "Hit", y: userInfo.stats.fairways_hit },
  { x: "Left", y: userInfo.stats.fairway_missed_left },
];
```

Figure 61 - Pie chart values initialisation

Shown in Figure 61 above is how I made the data to be passed into these pie charts. This was straight forward and required no loops, so I just initialised them as normal arrays. I then fed these arrays in to VictoryPie components.



```
else if (stat === "Score Distribution") {
  return (
    <VictoryPie
      width={350}
      theme={VictoryTheme.material}
      data={statsPie}
      animate={{
        duration: 2000,
      }}
    />
  );
} else if (stat === "Fairways Hit") {
  return (
    <VictoryPie
      width={400}
      theme={VictoryTheme.material}
      data={fairwaysPie}
      startAngle={90}
      endAngle={-90}
      style={{ labels: { fill: "black" } }}
      labels={({ datum }) =>
        `${Math.round((datum.y / totalFairways) * 100)}%`
      }
      labelComponent={<VictoryLabel dy={0} dx={-10} />}
      animate={{
        duration: 2000,
      }}
    />
  );
```

Figure 62 - Pie Charts made with VictoryPie

As you can see in Figure 62 above, I made two pie charts for the Score Distribution and Fairways Hit stats. For the Fairways chart I decided to make the labels the overall percentage of each metric. To do this I got the overall number of fairway stats recorded and divided each metric by that total and multiplied it by one hundred inside a Math.round function to get a percentage to one decimal place. I also wanted to make it a semi-circle pie chart to better visualise the metrics of missing left, right or hitting the fairway. To do this I set the start angle to 90° and end angle -90°. This created a semi-circle effect. The result of these Pie Charts can be seen in Figure 63 below.

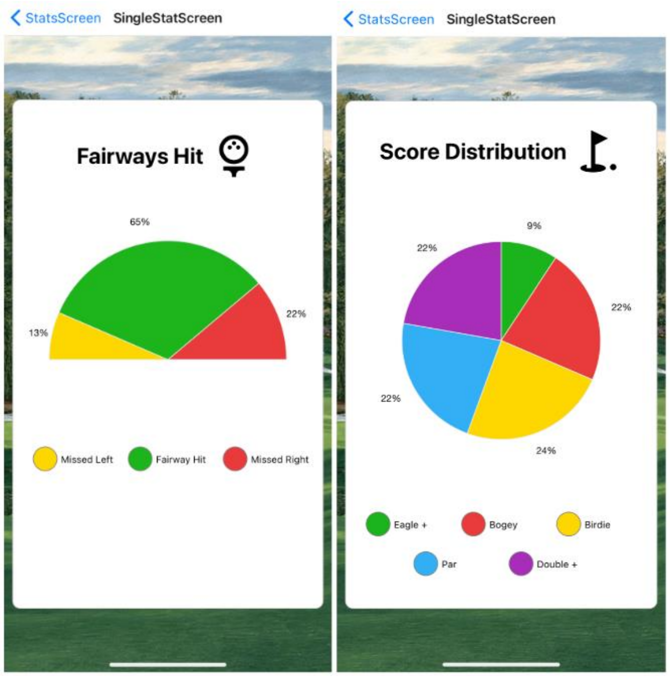


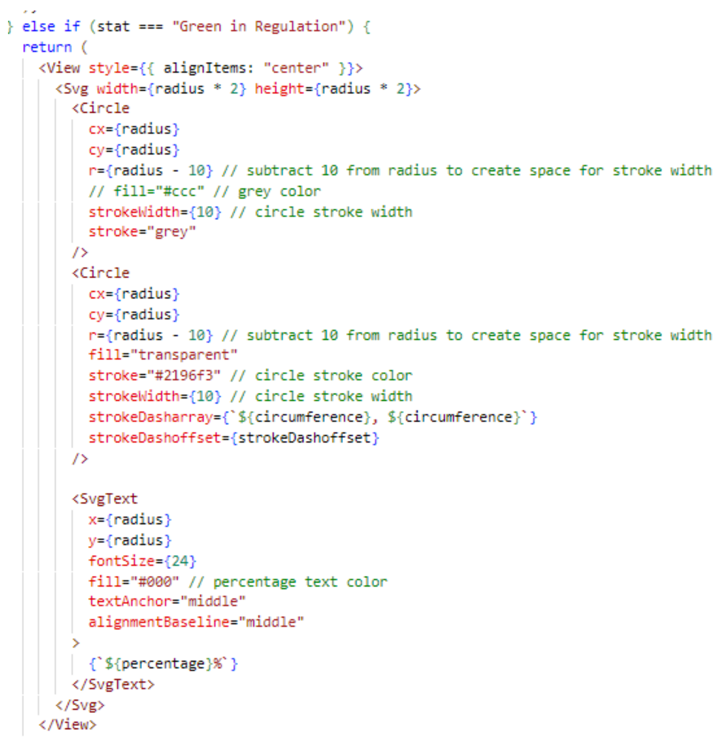
Figure 63 - Fairways and Score Distribution Pie Charts

Finally, I wanted to create a percentage circle to display the overall Green in Regulation percentage. To begin this process, I created an Svg with two circles and a text component inside. To make the percentage show up as a portion of a circle I needed to initialise a few values.

```
const percentage = Math.round(
  (userInfo.stats.greens_hit / (userInfo.stats.rounds_played * 18)) * 100
);
const radius = 100; // circle radius
const circumference = radius * 2 * Math.PI; // circle circumference
const strokeDashoffset = circumference - (percentage / 100) * circumference; // stroke dash offset
```

Figure 64 - Values need to create percentage circle

As you can see in Figure 64 above, I created four values. The percentage was the overall Green in Regulation percentage I wished to display, I made this by dividing the total number of greens hit by the user by the number of holes they have played and multiplied that number by 100 inside a Math.round function to get a percentage value correct to one decimal place. The next value was the radius of the circle which controls how large the circle is. Then the circumference which is calculated by 2πr. Then the strokeDashOffset which was calculated by taking the percentage of the circumference I wanted to show away from the overall circumference. I then plug these values into the circles.



```
} else if (stat === "Green in Regulation") {
  return (
    <View style={{ alignItems: "center" }}>
      <Svg width={radius * 2} height={radius * 2}>
        <Circle
          cx={radius}
          cy={radius}
          r={radius - 10} // subtract 10 from radius to create space for stroke width
          // fill="#ccc" // grey color
          strokeWidth={10} // circle stroke width
          stroke="grey"
        />
        <Circle
          cx={radius}
          cy={radius}
          r={radius - 10} // subtract 10 from radius to create space for stroke width
          fill="transparent"
          stroke="#2196f3" // circle stroke color
          strokeWidth={10} // circle stroke width
          strokeDasharray={`${circumference}, ${circumference}`}
          strokeDashoffset={strokeDashoffset}
        />

        <SvgText
          x={radius}
          y={radius}
          fontSize={24}
          fill="#000" // percentage text color
          textAnchor="middle"
          alignmentBaseline="middle"
        >
          {`${percentage}%`}
        </SvgText>
      </Svg>
    </View>
```

Figure 65 - Circle creation

Figure 65 above shows the return value for the Green in Regulation stat. I make two circles because I wanted to lay the percentage-based circle over a full circle, in order to make a complete looking circle with a highlighted portion instead just a partial circle. The result can be seen in Figure 66 below.

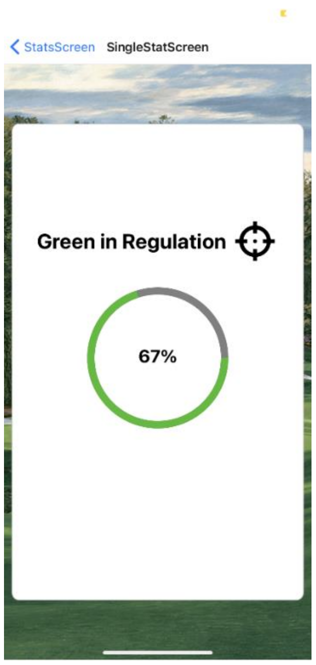


Figure 66 - Green in Regulation percentage circle

Now that I had all the main score entering and stat viewing functionality the next step was to work on the overall UI the application.

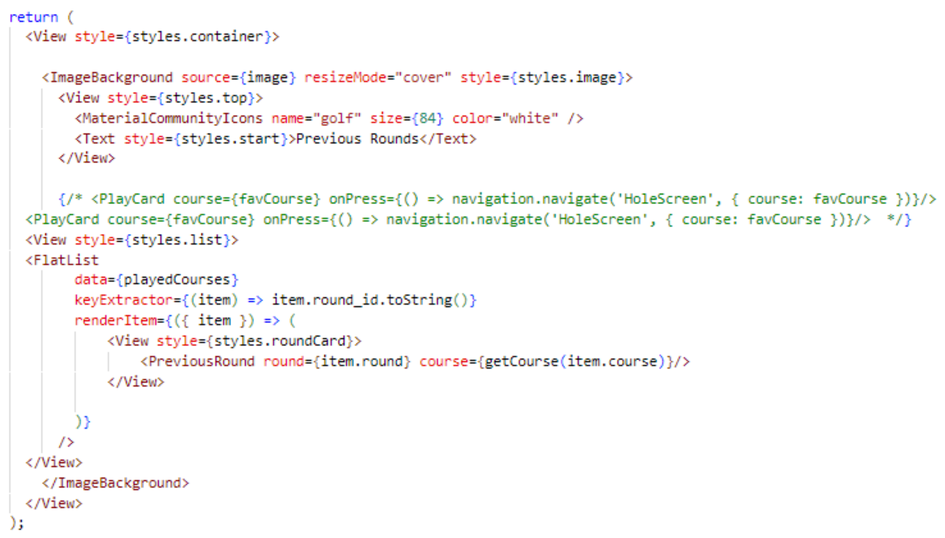
5.11 Sprint 8

5.11.1 Goal

The goal for Sprint 8 was to implement a Previous Rounds Screen and Favourite Courses screen.

5.11.2 Item 1 – Add Previous Rounds Screen

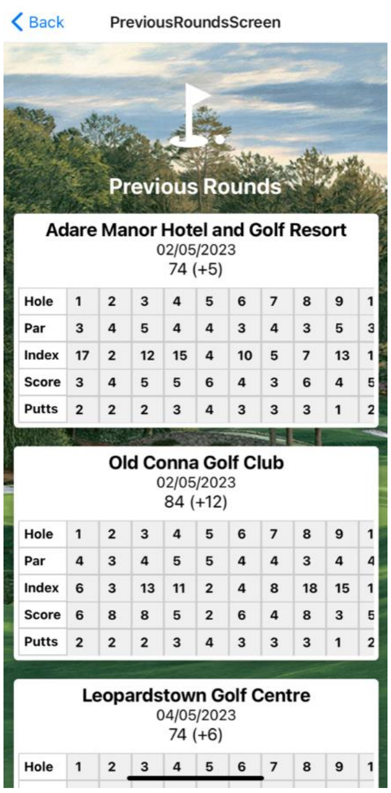
To implement the Previous Rounds Screen I first created a component like the one in the Submit Round Form. The only difference being that I made it take in an entire round from the user's played_courses array as opposed to just the form filled out by the user when entering their round. This gave me more information about the round to display such as the date it was played. In the Previous Rounds Screen I was then able to feed the users played_courses array into a FlatList and display the Previous Round Component.



```
return (
  <View style={styles.container}>

    <ImageBackground source={image} resizeMode="cover" style={styles.image}>
      <View style={styles.top}>
        <MaterialCommunityIcons name="golf" size={84} color="white" />
        <Text style={styles.start}>Previous Rounds</Text>
      </View>

      {/* <PlayCard course={favCourse} onPress={() => navigation.navigate('HoleScreen', { course: favCourse })}/>
  <PlayCard course={favCourse} onPress={() => navigation.navigate('HoleScreen', { course: favCourse })}/>  */}
  <View style={styles.list}>
  <FlatList
        data={playedCourses}
        keyExtractor={(item) => item.round_id.toString()}
        renderItem={({ item }) => (
            <View style={styles.roundCard}>
                <PreviousRound round={item.round} course={getCourse(item.course)}/>
            </View>

        )}
      />
  </View>
    </ImageBackground>
  </View>
);
```

Figure 67 - Previous Rounds Screen Code

Shown in Figure 67 above is the return value for the Previous Rounds Screen, here you can see the FlatList which returns a PreviousRound component. This component takes in the round and the course object which I retrieve with the course's ID by calling a getCourse function which searches for the courses ID in the full list of courses. The result of the Screen can be seen below in Figure 68.



Previous Rounds

Adare Manor Hotel and Golf Resort
02/05/2023
74 (+5)

Hole	1	2	3	4	5	6	7	8	9	1
Par	3	4	5	4	4	3	4	3	5	3
Index	17	2	12	15	4	10	5	7	13	1
Score	3	4	5	5	6	4	3	6	4	5
Putts	2	2	2	3	4	3	3	3	1	2

Old Conna Golf Club
02/05/2023
84 (+12)

Hole	1	2	3	4	5	6	7	8	9	1
Par	4	3	4	5	5	4	4	3	4	4
Index	6	3	13	11	2	4	8	18	15	1
Score	6	8	8	5	2	6	4	8	3	5
Putts	2	2	2	3	4	3	3	3	1	2

Leopardstown Golf Centre
04/05/2023
74 (+6)

Hole	1	2	3	4	5	6	7	8	9	1

Figure 68 - Previous Rounds Screen

The Screen shows every round the user has played along with the course they played, their score and the date they played.

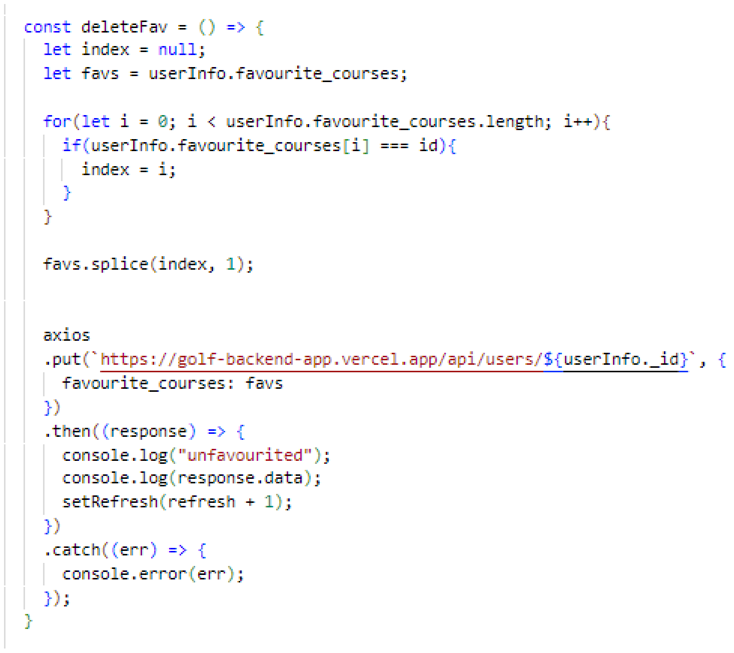
5.11.3 Item 2 – Favourite Courses Screen

I then began implementing the Favourite Courses Screen. I first had to implement functionality for favouriting a course. To do this I created a button that would be displayed on an individual course's Show Screen. I decided to make the button similar to a follow button you might see on other apps such as Twitter and Instagram. When pressed, the button calls an axios put request which adds the course's ID to the user's favourite_courses array.

```
const makeFav = () => {
  let favs = userInfo.favourite_courses;
  favs.push(id);
  axios
  .put(`https://golf-backend-app.vercel.app/api/users/${userInfo._id}`, {
    favourite_courses: favs
  })
  .then((response) => {
    console.log("favourited");
    console.log(response.data);
    setRefresh(refresh + 1);
  })
  .catch((err) => {
    console.error(err);
  });
}
```

Figure 69 - makeFav function

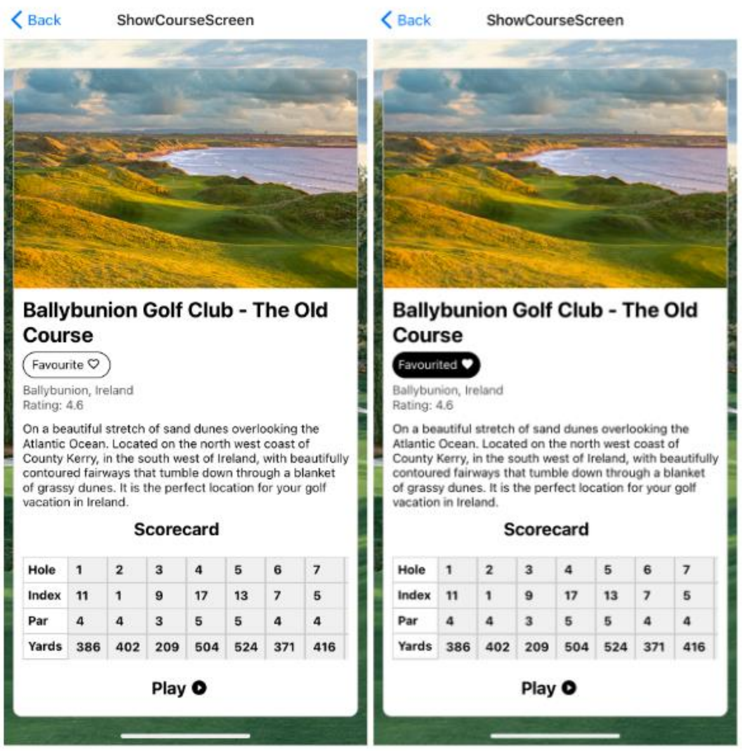
Figure 69 above shows the makeFav function. The favourite button initially displays an outlined button with the word favourite and when the course's ID is contained in the user's favourite_courses array changes to a filled in black button with the word favourited in it. When the favourited button is pressed an axios request removing the course from the user's favourites is called. This function is displayed in Figure 70 below.



```
const deleteFav = () => {
  let index = null;
  let favs = userInfo.favourite_courses;

  for(let i = 0; i < userInfo.favourite_courses.length; i++){
    if(userInfo.favourite_courses[i] === id){
      index = i;
    }
  }

  favs.splice(index, 1);


  axios
  .put(`https://golf-backend-app.vercel.app/api/users/${userInfo._id}`, {
    favourite_courses: favs
  })
  .then((response) => {
    console.log("unfavourited");
    console.log(response.data);
    setRefresh(refresh + 1);
  })
  .catch((err) => {
    console.error(err);
  });
}
```

Figure 70 - deleteFav function

The function searches for the index of the favourited course and uses the splice method to remove it. I then push this updated array back to the database. The result of these buttons can be seen below in Figure 71.



Ballybunion Golf Club - The Old Course

Ballybunion, Ireland
Rating: 4.6

On a beautiful stretch of sand dunes overlooking the Atlantic Ocean. Located on the north west coast of County Kerry, in the south west of Ireland, with beautifully contoured fairways that tumble down through a blanket of grassy dunes. It is the perfect location for your golf vacation in Ireland.

Scorecard

Hole	1	2	3	4	5	6	7
Index	11	1	9	17	13	7	5
Par	4	4	3	5	5	4	4
Yards	386	402	209	504	524	371	416

Ballybunion Golf Club - The Old Course

Ballybunion, Ireland
Rating: 4.6

On a beautiful stretch of sand dunes overlooking the Atlantic Ocean. Located on the north west coast of County Kerry, in the south west of Ireland, with beautifully contoured fairways that tumble down through a blanket of grassy dunes. It is the perfect location for your golf vacation in Ireland.

Scorecard

Hole	1	2	3	4	5	6	7
Index	11	1	9	17	13	7	5
Par	4	4	3	5	5	4	4
Yards	386	402	209	504	524	371	416

Figure 71 - Show Course Screens with favourite buttons

The next step was to implement the Favourite Courses Screen to display all of a user's favourite courses. To create this page, I decided to reuse the Play Card components for the courses. I had to set an array with all the user's favourite courses, to do this I created a getFavs function which was called in the axios request getting every course in the database. The function reads in the response from the axios call as a parameter and loops through each course checking its ID. If the ID of a course matches one in the user's favourite_courses array, it is added to the array. This getFavs function can be seen below in Figure 72.

```
const getFavs = (courses) => {
  let favArray = []
  if(courses){
      for(let i = 0; i < userInfo.favourite_courses.length; i++){
          for(let j = 0; j < courses.length; j++){
              if(courses[j]._id === userInfo.favourite_courses[i]){
                  favArray.push(courses[j])
                  console.log(courses[j].name)
              }
          }
      }
      // setTimeout(1000, getFavs);
      setCoursesList(favArray)
  }


}
```

Figure 72 - getFavs function

Once I had this array of all the courses a user had favourited, I fed it into a FlatList which displayed a Play Card for each course. The result of which can be seen below in Figure 73.

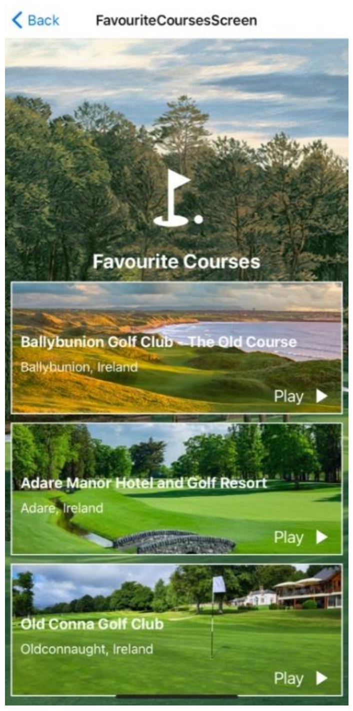


Figure 73 - Favourite Courses Array

The Favourite Courses Screen was now complete and so were all the Screens I planned to implement.

5.12 Conclusion

In this chapter I outlined the techniques and methodology that went into implementing my application. The implementation process was carried out over eight weekly sprints using the SCRUM Development Process. Each week I would implement several items and meet with my supervisor to review the progress I made. We would then decide upon the next set of items to complete for the next week. This process gave my work a nice structure and provided manageable goals to complete each week. In this chapter discussed my eight weekly Sprints outlining the goal every week as well as describing how I implemented each item I needed to complete. I went into detail explaining the code I wrote for each aspect of my program and displayed images of the end result of each component and screen added to the application. Overall the implementation process of my application was a success, I was able to implement every feature I had set out at the beginning of my project.

6 Testing

6.1 Introduction

This chapter describes the testing process that has been undertaken for the application. I will present this chapter in two sections:

1. Functional Testing
2. User Testing

Functional testing is a type of software testing whereby the system is tested against the functional requirements. The app is tested by looking to see if the actual output for a given input corresponds with the expected output. The tests are based on the requirements for the app. The results of functional testing can indicate if a piece of software is functional and working, but not if the software is easy to use. User testing aims to find out whether a piece of software is easy and intuitive for the user.

6.2 Functional Testing

This section describes the functional tests which were carried out on the app. These functional tests can be categorised as:

- Navigation
- Calculation
- CRUD

For this section of testing I will be using the Black Box Testing technique. This means that I will only be interested in whether the actual output of my test matches the expected output.

6.2.1 Navigation

Test No	Description of test case	Input	Expected Output	Actual Output	Test Result	Comment
1	Sign up with valid credentials	Name: Test Email: test@test.test Password: testtest	Account created and user is authenticated	Account is created and user is authenticated	Pass	
2	Log in with valid credentials	Email: test@test.test Password: testtest	User is authenticated and is brought into app	User is authenticated and is brought into app	Pass	

3	Sign up with invalid credentials	Name: tes Email: test'sa Password: test	User is given relevant input error feedback	User is given relevant input error feedback	Pass	Feedback is given based on which credential is invalid
4	User is authenticated and presses courses tab	Press Courses tab	User is brought to courses screen	User is brought to courses screen	Pass	
5	User is authenticated and presses play tab	Press Play tab	User is brought to play screen	User is brought to play screen	Pass	
6	User is authenticated and presses me tab	Press Me tab	User is brought to Me screen	User is brought to Me screen	Pass	
7	User presses Stats button in Me tab	Press Stats button	User is brought to Stats Screen	User is brought to Stats Screen	Pass	
8	User presses any individual Stat on Stats Screen	Press individual Stat	User is brought to Individual Stats Screen	User is brought to Individual Stats Screen	Pass	
9	User presses Favourite Courses Button in Me Screen	Press Favourite Courses Button	User is brought to Favourite Courses Screen	User is brought to Favourite Courses Screen	Pass	
10	User presses Previous Rounds Button in Me Screen	Press Previous Rounds Button	User is brought to Previous Rounds Screen	User is brought to Previous Rounds Screen	Pass	
11	User presses on a course in courses screen	Press on course card in courses screen	User is brought to Show Course screen for specific course	User is brought to Show Course screen for specific course	Pass	
12	User presses play in Show Course screen	Press play button in Show Course Screen	User is brought to specific course's Hole Screen	User is brought to specific course's Hole Screen	Pass	
13	User presses Play on course in Play Screen	Press play button on course card in Play Screen	User is brought to specific course's Hole Screen	User is brought to specific course's Hole Screen	Pass	
14	User presses sign out button in Me Screen	Press Sign Out button in Me Screen	User is unauthenticated and brought to Welcome Screen	User is unauthenticated and brought to Welcome Screen	Pass	

6.2.2 Calculation

Test No	Description of test case	Input	Expected Output	Actual Output	Test Result	Comment
1	Calculating total score to par as user enters their scores	Score on hole is entered	Total score to par is calculated	Total score to par is calculated	Pass	
2	Calculating total score when user submits a round	Round is submitted	Total score for round is stored	Total score for round is stored	Pass	
3	Calculating word value for each hole in round when user submits	Round is submitted	Word values for each hole entered is stored	Word values for each hole entered is stored	Pass	
4	Calculating number of putts taken during a round	Round is submitted	Number of putts taken for the round is stored	Number of putts taken for the round is stored	Pass	
5	Calculating number of fairways hit during a round	Round is submitted	Number of fairways hit for the round is stored	Number of fairways hit for the round is stored	Pass	
6	Calculating number of greens hit during a round	Round is submitted	Number of greens hit for the round is stored	Number of greens hit for the round is stored	Pass	

6.2.3 CRUD

Test No	Description of test case	Input	Expected Output	Actual Output	Result	Comment
1	User registering	User enters valid credentials into sign up form	User is added to database	User is added to database	Pass	
2	User fills out a round	User enters scores for 18 holes	Round is added to user's played_courses array in database	Round is added to user's played_courses array in database	Pass	
3	User views all courses in the Courses Screen	User goes to Courses Screen	User can see all courses from the database	User can see all courses from the database	Pass	
4	User favourites a course	User presses the favourite button on the show course screen	Course's ID is added to user's favourite_courses array in database	Course's ID is added to user's favourite_courses array in database	Pass	
5	User unfavourites a course	User presses the favourited button on the show course screen	Course's ID is removed from user's favourite_courses array in database	Course's ID is removed from user's favourite_courses array in database	Pass	
6	User views previous rounds	User presses the Previous Rounds button in the Me Screen	User can see all previous rounds played	User can see all previous rounds played	Pass	
7	User views favourite courses	User presses the Favourite Courses button in the Me Screen	User can see a list of all favourite courses	User can see a list of all favourite courses	Pass	
8	User views their stats	User presses the stats button in the Me Screen and chooses a specific stat to view	User can see their chosen stat metrics in the Individual Stats Screen	User can see their chosen stat metrics in the Individual Stats Screen	Pass	

6.2.4 Discussion of Functional Testing Results

The functional testing I carried out was a success. All navigation items passed their tests, this was very pleasing, I tested each item as I was implementing them individually so I was confident that each test would pass during the overall functional testing review. The same can be said for the calculation and CRUD components of my testing. The calculation tests were all successful, five out of six items are operated by the same submit round function that is called when a user submits their round to the database. The CRUD tests were all successful, my application does not use any delete functions, but it does utilise put functions to add and remove items from a user in the database.

6.2.5 User Testing

I carried out User Testing in person, with users next to me. This was because my application could not easily be hosted and used remotely. I asked four users to carry out a list of tasks and recorded results on paper.

Log in/Sign up

The first test conducted during the user testing was asking users to register and log in. For all users registering caused no trouble with 100% of users agreeing that registering was easy to do. I did not tell users the form validation rules, my app does give feedback however when users enter invalid credentials and enforces proper entries into each field. Users also receive feedback from the app when invalid log in credentials are inputted.

Search For and Pick a Favourite Course

The second task conducted during the user testing was to ask users to search for Adare Manor Golf Club and add it to their favourites. The results were pleasing with 80% of users having no issue finding and adding the course. One user commented that they were unsure how to add a course to their favourites. The favourite button was not extremely obvious to the user, this is something I will investigate, with a larger sample size this problem might be minimised.

Enter and Submit a Round at Old Conna Golf Club

The third task conducted during the user testing was to ask users to start and submit a round at Old Conna Golf Club. Results were positive with 100% of users being able to find the hole screen and begin entering data. It would have been a more useful test if users were able to be out on the course playing it in real time, this was an unfortunate reality of my testing. Users commented that entering a round of eighteen holes while sitting at a desk is quite monotonous. Users did experience no issues in doing so however so results were pleasing.

6.3 Conclusion

In conclusion I carried out Functional Testing and User Testing on my application. I set out to ensure that my app successfully implemented all my functional and non-functional requirements. I believe I achieved my goal, functional testing was a success with all tests passing in every category. User testing was successful also with only some minor UI items that were brought up. I will look to improve the overall UI of my application with these criticisms in mind.

7 Project Management

7.1 Introduction

This chapter describes how the project was managed and how I worked with my supervisor to create and implement my project. It shows the phases of the project, going from the project idea through the requirements gathering, the specification for the project, the design, implementation and testing phases for the project. It also discusses Trello, GitHub and project member's journals as tools which assist in project management.

7.2 Project Phases

In this section I will describe all the different project phases that went into developing my application from start to finish.

7.2.1 Proposal

The first phase of the project was to come up with an idea and propose it to my supervisor. I had an idea that I wanted to make something involving golf as I am passionate about it. Initially I thought I would be making a type of golf community platform which would have a big social element. I originally made the decision to implement this as an Angular.js web application. I was then advised by my supervisor that although an Angular.js project would be different to what we had used before, a mobile application would be a much better route to go down. I was happy to go along with this idea as it gave me the chance to work with React Native which is a framework that I had great interest in using.

As the technology I was going to use had changed, I decided to modify the idea for the application. I thought as it was a mobile app, it would be a good idea to have something that people could use when they are the course. I produced the idea for a golf score tracking app, where users could enter scores and stats about their rounds and track them over time. I proposed this idea to my supervisor, and he approved. I then began gathering my requirements for the application.

7.2.2 Requirements

The second phase of the project was to gather the requirements for my application. The first step in this process was to study some applications that were similar to mine. This allowed to me see how these types of apps are constructed and gain inspiration from them. This process also uncovered aspects of these apps which I didn't like or felt I could improve upon. The next step was to conduct some interviews and surveys on prospective users. I asked questions about their experience using golf applications and metric tracking apps in general such as step counters etc. I was able to get information from these potential users about their favourite and least favourite aspects of these types of applications, along with ideas of features they might like to see added. This process gave me some useful information and formed a wish list of features I would look to implement.

The next step was to begin creating some models to create a clearer picture of the type of people that would be using the app, and how they would interact with it. I created some proto personas of imaginary but realistic potential users each with their own desires and frustrations. This gave me a look at the demographic of users I wanted to reach. I then created a Use Case Diagram to visually gain an understanding of the different operations that users would be able to access while utilising the application. After this I was able to create a full list of functional and non-functional requirements for my application. The functional requirements comprised of technical functions that I needed to implement, while the non-functional requirements consisted of items that are not vital to the pure functionality of the application but were important to the overall ergonomics and user experience.

7.2.3 Design

The third phase of the project was the design phase, in which I looked at both the program design and the user interface design. The first step in this process was to outline the different technologies I would be using, and how they were structured. I studied coding standards and conventions to ensure I laid out my project files in a well-structured and organised manner. This aided fluidity while coding as I knew where everything should be put, and where I could access any files that I needed.

I then created an Application Architecture Block Diagram to get a big picture view of the different sides of the application I was implementing. This clarified the request and response processes of the client and server along with the accessibility of the database through the server. I then created an Entity Relationship Diagram which outlined how the different data tables in my database were structured and how they would relate to each other.

Process design was the next step and to aid me in this was the creation of two more diagrams. The first of which was a Sequence Diagram which illustrated how users would be interacting with the back end of the application. The second diagram was a Flow Chart which enabled me to visualise the different actions and decisions users will make as they go through the application. This helped me understand the flow of the app and identify any bottlenecks or issues that might crop up.

I then moved on to user interface design. I used Figma to create wireframes of my application. I made wireframes for each individual screen to give myself a visual goal to work towards while implementing the application. I also decided upon a style guide to work from so that I had a consistent and well thought out design for each component and element within the app. This process enabled me to develop a nice-looking application which had a consistent theme throughout.

7.2.4 Implementation

The fourth phase of the project was the implementation of my application. This process was carried out over eight weekly sprints using the SCRUM Development Process. Each week I would implement a number of items and meet with my supervisor to review the progress I made. We would then decide upon the next set of items to complete for the next week. This process gave my work a nice structure and provided manageable goals to complete each week.

7.2.5 Testing

The fifth phase of the project was carrying out functional and user testing on my application. I began this process by utilising the Black Box Testing technique to put my application through a series of tests regarding its overall functionality. With these tests I was only concerned that actual output matched expected output. These tests were successful and showed I had achieved all my functional requirements. I then carried out user testing in person with four prospective users, I asked them to carry out a series of tasks and recorded their results and feedback. Overall results were successful, users gave positive feedback on the application's ease of use and design.

7.3 SCRUM Methodology

The Scrum methodology revolves around a particular set of practices and roles incorporated in the development of software. A Scrum takes place in small periodic sections called Sprints, which are generally anywhere from one to four weeks long and conclude with feedback and reflection. All Sprints are their own separate entity that performs at least one tangible task that can be reviewed with ease.

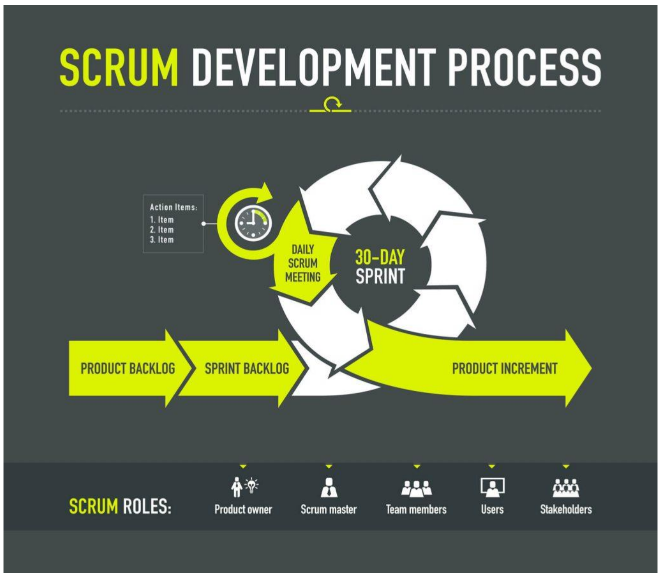


Figure 74 – SCRUM Development Process

As Figure 74 above shows, the process is designed to complete a list of deliverables in a brief period of time with evident accountability. The client is the one who prioritises objectives considering value and cost. The market requires quick, agile and flexible development of products to stay up to date with consumer needs and not be left in the dust of the rapidly developing tech industry. The scrum methodology is a terrific way to ensure these needs are met.

There are three main roles in a Scrum team:

1. The Scrum master is the leader of the team and is in charge of keeping the scrum up to date, training, coaching and mentoring the team where needs be.
2. The Product owner is the representative of the stakeholders and customers that will use the software. Their main objective is maximising return on investment. They communicate the vision of the product to the team and prioritise the Product Backlog regularly.
3. The Team is the group of professionals with the required skills to translate the vision of the product to reality.

The implementation phase for this project consisted of eight sprints in total. Each sprint took place over a period of 1 week. The sprints were highly successful and helped ensure I stayed on track with my implementation. The requirements for the application were listed in a product backlog. Each item on the product backlog was broken down into a series of tasks which formed a sprint.

7.4 Project Management Tools

I used one main Project Management Tool while going through the development process of my application.

7.4.1 GitHub

GitHub is a provider of internet hosting and version control. I utilised it to manage my code for both the front end and back end of my project. I created a different repository for each end and made regular commits to them as I altered and added code throughout the implementation of my application. I used Vercel to host the back end of my project and this was linked to my GitHub repository, this allowed my server to be updated as soon as I committed code. I made comments with every commit which enabled me to keep a record of the work I had done and gave me a nice timeline to look back on when writing up my report.

7.5 Reflection

In this section I will reflect upon all the various aspects of my project.

7.5.1 Your views on the project

I felt the project went very well overall from my perspective. The process of coming up with an idea for an application and seeing it through to the end was very rewarding. Going through the steps of creating a fully functional mobile application has provided me with crucial experience for my future endeavours. I enjoyed implementing the application myself as I had to keep myself accountable and disciplined in order to stay on track. It also made me have to troubleshoot a lot by myself, this really made me dive deep into understanding how and why a vast array of errors happen and how to fix them.

7.5.2 Completing a large software development project

It was a very good experience getting a taste of what a large software development project would be like. Although it was only myself involved in the development of this specific project, I still believe that the methodology I followed, meeting with a supervisor and providing deliverables weekly, would be seamlessly scaled up to whichever level required.

7.5.3 Working with a supervisor

I felt it was a good experience working with a supervisor. It was always helpful to remain accountable for my deadlines and have someone to ask questions about anything that came about during the process of my project.

7.5.4 Technical skills

I believe my technical skills improved a huge amount throughout the development of my project. I went into this project with little to no experience creating mobile applications especially using React Native. I feel like I have become quite proficient using the framework, I didn’t use a huge number of libraries to implement my app, because of this I did a lot of work from scratch and gained a nice understanding of the capabilities of the base React Native framework. I feel confident in my ability now using it and hope to continue using it professionally in the future. I also continued enhancing my skills with Express and MongoDB which I have now used for several projects.

7.5.5 Further competencies and skills

I think in relation to developing mobile applications I would like to gain more experience with React Native and other frameworks such as Flutter. I would also like to improve my skills when it comes to database management and automated testing. I think those skills would make me very employable and give me great confidence entering the tech industry.

7.6 Conclusion

In this chapter I described all of the different phases of my golf score tracking application as well as what tools and methodology I used to carry out my project. I also reflected upon the various aspects of the project and what I gained from the experience. I learned a great amount about developing mobile applications, having a supervisor and developing software. I also realise there are several things that would be useful for me to do when it comes to learning different frameworks and languages for developing software. These technologies would help me greatly in the future when trying to enter the tech industry.

8 Conclusion

The goal of this project was to create a Golf Score Tracking App that me, my friends and many other golfers out there could use. The game of golf is growing rapidly, and players are always looking for new ways to enter their rounds and keep track of their stats. Players have historically entered their scores on physical paper scorecards, but they become damaged, lost and forgotten about. This app aims to provide golfers with a new easy way to keep track of their scores, stats and previous rounds all in one place and accessible at the click of a button. I myself am a keen golfer and I am always trying to improve my game. Keeping track of your stats over time lets you know your strengths and weaknesses and gives your practice a focus. Chapter 2 of this document is a literature review comparing different JavaScript Frameworks and how they should be considered when building an application of this type.

For technologies I essentially used a MERN (MongoDB, Express, React and Node) stack, however I used React Native instead of just pure React. MongoDB is a NoSQL database which stores my application's data in a JSON like format. I chose MongoDB because of its seamless compatibility with React Native and Express. Express.js is a back end Node.js framework, I used it to build the server for my application. I chose Express because I had previous experience using it and felt it was the most appropriate back end framework to use. React Native is a JavaScript framework built on the functionality of React, which is one of the most popular JavaScript frameworks in the world used for developing web applications. React Native allows developers to use this framework in conjunction with native platform functionality. It allowed me to write one set of code and deploy it across numerous platforms, these include iOS, Android and Web. The main platform I deployed and tested it on was iOS, I was able to run a local server on my PC and connect to it on my phone via the ExpoGo app. This allowed me to view, utilise and edit the app in real time. Other technologies that could have been considered were Flutter and Swift however for the purpose of my application and the hardware I had access to, React Native was the best choice.

I researched JavaScript Frameworks and how they should be considered when building an online community website. This process gave me great insight into the factors needed to be considered when choosing a framework to develop an application. My findings led me to choose React Native to develop a mobile application.

I went through a number of steps to gather the functional and non-functional requirements for my application. I first looked at similar applications to mine to gain inspiration and seek any faults or flaws that I could improve upon. Secondly, I conducted interviews and sent out a survey to gather information from prospective users about what they liked or disliked about any golf applications they used, as well as any new ideas they might like to see added to such an app. I then created proto personas of some of the different types of users I may have. This outlined a range of potential users and what their needs and wants may be. I also created a Use Case Diagram to gain a visual understanding of all of the different operations users would undertake while using the app. I was then able to develop a full list of all of the functional and non-functional requirements for my application.

The first step in this process was to sketch out the various technologies I would be employing and how they would be organized. I researched coding standards and conventions to ensure that my project files were well-structured and organized. I then created an Application Architecture Block Diagram to get a bird's-eye view of the various aspects of the application I was developing. This clarified the client and server request and response processes, as well as the server's access to the database. I then created an Entity Relationship Diagram to show how the various data tables in my database were structured and how they would interact with one another. The next step was process design, which was aided by the creation of two more diagrams. The first was a Sequence Diagram, which depicted how users would interact with the application's back end. The second diagram was a Flow Chart, which allowed me to visualize the various actions and decisions that users would make as they navigated the application. This aided me in understanding the app's flow and identifying any bottlenecks or issues that might arise. I then moved on to designing user interfaces. Figma was used to create wireframes for my application. I created wireframes for each individual screen to provide myself with a visual goal to strive for while developing the application. I also created a style guide to follow so that I could create a consistent and well-thought-out design for each component and element within the app. This process enabled me to create a visually appealing application with a consistent theme throughout.

Implementation was carried out using the SCRUM Development Process, this process was carried out over eight weekly sprints. I would implement a number of items each week and meet with my supervisor to review my progress. We'd then decide on the next set of tasks for the following week. This procedure gave my work a nice structure and set manageable goals for each week.

The testing process involved conducting functional and user testing on my application. I started this process by using the Black Box Testing technique to run a series of tests on my application's overall functionality. I was only concerned with the actual output matching the expected output in these tests. These tests were successful, demonstrating that I had met all of my functional requirements. I then conducted in-person user testing with four prospective users, asking them to complete a series of tasks and recording their results and feedback. The results were positive, with users complimenting the application's usability and design.

My project management was aided by the SCRUM methodology where I carried out 8 weekly sprints. I met with my supervisor every Wednesday to discuss and display my progress, we then came up with a new set of items for me to implement for the next week. I used GitHub for my version control, I had repositories for both the front end and back end of my application. I made regular commits along with comments every time I made changes and implemented functionality.

I learnt a great deal throughout the development of my application. I gained a huge amount of experience using React Native to create a functional mobile application. I had very little experience working with mobile applications before and I now feel very confident in my ability in this area. I continued my experience using Express and MongoDB which are extremely useful tools to have proficiency in.

The application has a lot of room for growth and future development. I would like to add more modes to the playing functionality, for example a popular format for playing golf is Stableford. In this format players are awarded points for each hole based on their net score which is calculated by the golf handicap system. This feature would be a very useful tool to add as a lot of competitive golf in most clubs is played in this format. Another feature I would like to add is a multiplayer aspect where users can enter scores for more than one person at a time. This would go along well with the usual way of competitive golf scorecard keeping where each member of the group keeps track of each other’s score for verification purposes. This can also just be used casually where only one member of the group needs to keep score for everyone. Overall, there are countless features that could be added to the app, however I wanted to keep my application focused on one main aspect for the purposes of quality and the timeframe we had.

References

Davies, A., Mueller, J. (2020). Usability Testing and Deployment. In: Developing Medical Apps and mHealth Interventions. Health Informatics. Springer, Cham. Retrieved from: https://doi.org/10.1007/978-3-030-47499-7_7

Elar Saks (2019). JavaScript Frameworks: Angular vs React vs Vue. Retrieved from: https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf

Flavian, C., Gurrea, R. and Orús, C. (2009), "Web design: a key factor for the website success", *Journal of Systems and Information Technology*, Vol. 11 No. 2, pp. 168-184. Retrieved from: https://doi.org/10.1108/13287260910955129

Golf Business News. (n.d.). *Golf participation returns to growth in GB & Ireland in 2022*. GolfBusinessNews.com. https://golfbusinessnews.com/news/growing-the-game/golf-participation-returns-to-growth-in-gb-ireland-in-2022/

Hsiu-Fen Lin and Gwo-Gwuang Lee, (2007). Determinants of success for online communities: an empirical study. Retrieved from: https://doi.org/10.1080/01449290500330422

Jacob Amedie, (2015). "The Impact of Social Media on Society" *Pop Culture Intersections*. 2. https://scholarcommons.scu.edu/engl_176/2

Jensen, S.H., Møller, A., Thiemann, P. (2009). Type Analysis for JavaScript. In: Palsberg, J., Su, Z. (eds) Static Analysis. SAS 2009. Lecture Notes in Computer Science, vol 5673. Springer, Berlin, Heidelberg. Retrieved from: https://doi.org/10.1007/978-3-642-03237-0_17

Jeremy Keith (2009). DOM Scripting – Website Design with JavaScript and the Document Object Model. Retrieved from: https://doi.org/10.1007/978-1-4302-0062-8

Josefin Salomaa (2020). Evaluating JavaScript Frameworks. Retrieved from: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1461878&dswid=9256

M.J Taylor, J McWilliam, H Forsyth, S Wade (2002) , Methodologies and website development: a survey of practice, Information and Software Technology, Vol. 44, Issue 6, pp. 381-391. Retrieved from: https://doi.org/10.1016/S0950-5849(02)00024-1

Oliver K. Burmeister (2010). Website for Seniors: Cognitive Accessibility. International Journal of Emerging Technologies ad Society. Vol. 8, No, 2, pp: 99-113. Retrieved from: https://researchoutput.csu.edu.au/ws/portalfiles/portal/8778491/PostpubPID22728.pdf

Pano, A., Graziotin, D & Abrahamsson, P. (2018) Factors and actors leading to the adoption of a JavaScript Framework. Retrieved from: https://doi.org/10.1007/s10664-018-9613-x

Robert Plant (2004). Online Communities, Technology in Society, Vol.26, Issue 1, pp: 51-65. Retrieved from: https://doi.org/10.1016/j.techsoc.2003.10.005

Robert Satrom, (2007). Choosing the Right JavaScript Framework for Your Next Web Application. Retrieved from: https://www.telerik.com/docs/default-source/whitepapers/telerik-com/choose-the-right-javascript-framework-for-your-next-web-application_whitepaper.pdf

Sanjay Misra and Ferid Cafer (November 2012). The International Arab Journal of Information Technology, Vol. 9: Estimating Quality of JavaScript. Retrieved from: http://iajit.org/PDF/vol.9,no.6/3056-6.pdf

S. Delcev and D. Draskovic, (2018). "Modern JavaScript frameworks: A Survey Study," *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pp. 106-109, Retrieved from: https://ieeexplore.ieee.org/abstract/document/8448444

Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. (2007). Analysis of topological characteristics of huge online social networking services. In Proceedings of the 16th international conference on World Wide Web (WWW '07). Association for Computing Machinery, pp: 835–844. Retrieved from: https://doi.org/10.1145/1242572.1242685