Sports Advisor - Kotlin Android Application

Robert Donnelly

Evan Greaney

Steven Joyce

B.Sc.(Hons) in Software Development

May 10, 2021

Final Year Project

Advised by: Kevin O'Brien
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

Chapter 1

About this project

1.1 Abstract

This project is a Mobile Application for the Android Operating System. The programming Language used is Kotlin. The development Environment that the application is written in is IntelliJ IDEA Ultimate.
The app is a sport-based application that allows the user to find all golf courses in Galway. The user will be given the weather forecast over a 12 hour period, or current time conditions depending on what page they are on. The weather forecast will be specific to what golf course that the user wants to play at. It will give the user back the scored ratings for each hour to play on for that period of time.
The application will allow the user to access a MongoDB database to store and fetch personalized data. The stored data is your score for each hole and what par that hole was given by the user. The score is calculated by finding the par of the course and subtracting

1.2 Authors

For this Project, the authors are Robert Donnelly, Evan Greaney and Steven Joyce. All three authors are 4th Year Software development students, attending GMIT.

Chapter 2

Introduction

For our final year project, we were tasked with designing, developing, and deploying a project either individually or as a group. We decided to work together as a group in order to achieve our goals of creating an app that would assist as a companion to players for one sport with the hope of adding in more sports in the future. The sport we decided to focus on was the game of golf, in which the app would give players optimal playing days and times based on weather conditions as well as their preferred conditions for playing golf. The application would also allow players to save their score data for that day which can be accessed in a report or table.

2.1 Objectives

- Find a new programming language to learn independently
- Find a Methodology to help the development of our project
- Implement the methodology to create a project plan
- Find an architecture structure suitable to use
- Utilize an environment for stable version control
- Establish a well-structured base for the project
- Create a high-quality front-end
- Find and deploy an API which our app can utilize
- Create a server service for the Application.

- Allow for user data to be protected and appropriately stored
- Use of a Location service to find user's desired location
- Implement a login service which allows user to access their protected data
- Design the Apps software to successfully implement the apps intended functionality
- Deploy our methodologies style of testing to ensure quality standards are met

2.2 Chapters

2.2.1 Methodology

This chapter covers our implementation of the Agile Methodology used to collaborate as a team to structure our project schedule to take an incremental and iterative approach to the development of our Mobile Application.

2.2.2 Technology Review

In this Area of this Dissertation, we will cover all technologies used to develop this application, to design this application, and will give the reader an understanding of why these technologies were used.

2.2.3 System Design

The chapter on System Design highlights how this Mobile Application was developed using the technologies outlined in the Technology Review chapter. It will also outline and illustrate the System design that was imagined and conceived.

2.2.4 System Evaluation

The System Evaluation section of this Dissertation will cover the testing stage of the Application, where the finalised Application will be compared to the initial objectives of the Application set out by the Introduction. It will outline the faults and limitations of the technologies used to design and develop the project along with concepts for future advancement of the Application.

2.2.5 GitHub Repository

Link to GitHub Repository=`https://github.com/stevenJoyce/4thYearGroupProject`

1. **App Folder:**
   This folder contains the Files necessary to build the Application
2. **Dissertation Folder:**
   This is where the Dissertation can be run from Overleaf
3. **Dissertation PDF:**
   This is where the final version of the Dissertation can be viewed from
4. **ReadMe:**
   Gives a brief overview of what is contained within the GitHub Repository
5. **Presentation of Initial Concept:**
   This is a PowerPoint presentation designed to highlight the initial concepts and prototypes for the project
6. **TestingGuides Folder:**
   The folder contains the Gantt Chart - This is an excel file that displays the implementation of Agile Methodologies to design sprints of development and the test cases we used to test each sprint with default data to get back a generic result. The test cases were designed in Excel.
7. **.Ideas Folder:**
   This folder is where we ran and debugged our Application from in the IntelliJ IDEA IDE.
8. **Gradle Folders:**
   Using this folder, this is how the Application is configured using IntelliJ
9. **Wireframe Prototype:**
   In this folder we have a Justinmind prototype outlining the design of the application.

Chapter 3

Methodology

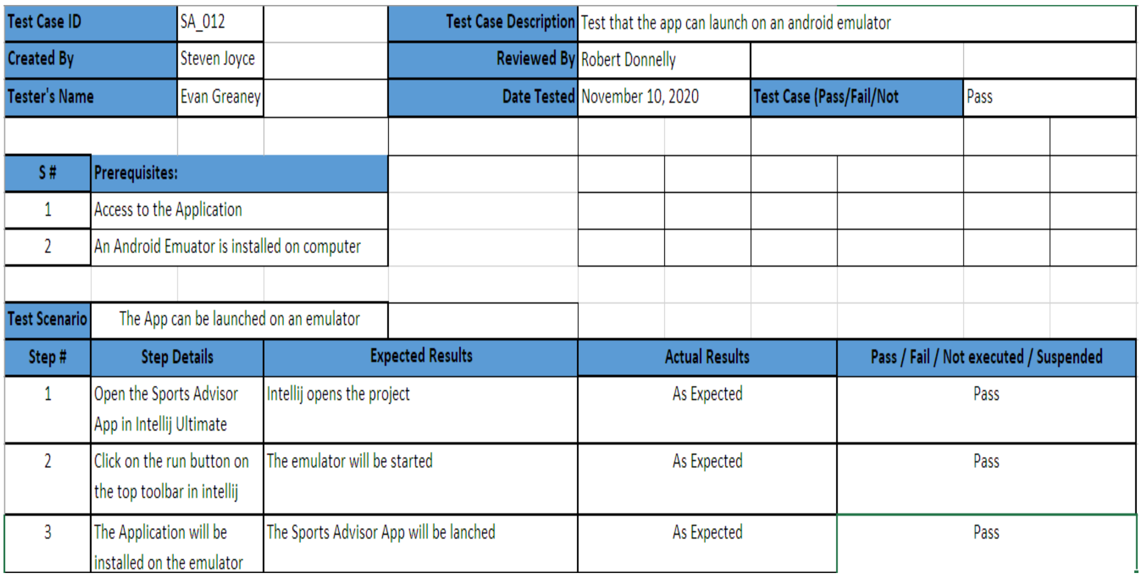
3.1 Approach

The Agile methodology is based on an iterative development approach that allows requirements and solutions to change throughout the development of the Application. It relies on constant communication between members of the team to successfully deliver a final product that has the capacity to evolve. Teams will undergo sprints that splits up a feature into smaller parts that are then given to a member of the team to work on in a short period of which makes the feature more manageable to deliver.

To implement this approach our team organized weekly meetings where we discussed our work's success or issues and how far we have gotten with our given sprint. This allowed us to collaborate as a team when issues occurred allowing us to keep on schedule. Our schedule was developed using a Gantt chart, this was the basis for all our sprints. The Gantt chart is spoken about in more detail in Chapter 6 - System Evaluation. Every step was assigned as a sprint which was assigned to each member to carry out. Each sprint had a test case that had to pass for us to proceed with development.

3.2 Testing

After each sprint, a set of tests were required to pass. If a test case failed to pass in any capacity we could not proceed with the next sprint. To test our application we used the Android SDK emulator extension included in the IntelliJ IDEA. The emulator we used was run on Android 9.0 (pie). The device that the emulator is simulated is the Google Pixel 3A mobile phone.



Test Case ID	SA_012	Test Case Description	Test that the app can launch on an android emulator		
Created By	Steven Joyce	Reviewed By	Robert Donnelly		
Tester's Name	Evan Greaney	Date Tested	November 10, 2020	Test Case (Pass/Fail/Not	Pass

S #	Prerequisites:
1	Access to the Application
2	An Android Emuator is installed on computer

Test Scenario	The App can be launched on an emulator

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open the Sports Advisor App in Intellij Ultimate	Intellij opens the project	As Expected	Pass
2	Click on the run button on the top toolbar in intellij	The emulator will be started	As Expected	Pass
3	The Application will be installed on the emulator	The Sports Advisor App will be lanched	As Expected	Pass

Figure 3.1: Test Case for successful Login

The image above illustrates a template test case we designed and implemented for testing the functionality of our application. The tests are designed to check each component of the App for correct implementation.

3.3 Collaborative Platforms

3.3.1 Version Control

The platform all team members were most familiar with was GitHub, it allows members to access project code versions and commit their contributions to one repository. Every team member has access to view and modifies the contents of the repository. All changes are logged and any commit can be accessed at all times, this helps with problem-solving as any issues that occur can be called back on to older commits that were functional if issues occur with the latest commit.

3.3.2 Communication Tools

Microsoft Teams

Microsoft Teams was used primarily to communicate with our supervisor. We chose this as it is linked to our student accounts and our supervisor requested us to use it as a way to communicate with him.

Microsoft Teams is a good platform for video calls as it can be accessed on multiple platforms such as an Internet browser, Mobile/Desktop Application. It allows users to share files and can sync to your google drive account(file hosting service) and use a calendar feature allowing users to organize meetings.

However it is not strongly suited for messaging as its forum feature can become cluttered and tedious to navigate, This is one of the main reasons why we decided to use Discord as our primary form of communication as a team.

Discord

Discord is a communication Application that has increased in popularity in recent years, especially considering the current pandemic. It runs on Desktop, Mobile, and browser platforms. This allowed us to always be in contact with each other if one team member's device was running into issues.

Discord allowed us to set up separate channels to sort our messages to relate to specific aspects of the Project. We could coordinate and plan different parts of the project in one place with no clutter, Each aspect of the project had a dedicated channel so that unrelated messages would bombard our message feed with irrelevant information.

For example, we had an ideas channel where we would jot down random ideas or inspiration for new features of the application, these messages would be separate from our session-planning channel which would strictly contain messages related to our sprint meetings. Due to this feature, it became our primary platform to communicate.

Whats App

We used Whats App on our mobile devices to organize our sprint meetings as a team. If for any reason a meeting needed to be held on short notice What's App was the most efficient way of coordinating a set time.

3.3.3 Literature Software

For the Development of this Dissertation we used Overleaf, we found this Latex text editor the effective way to write up due to our previous experience. The Documentation Overleaf provides is well structured which helped us with this aspect of the project. After each person wrote different parts of the dissertation, it was then committed to GitHub, which helped with making sure every member of the team had the appropriate version of the Dissertation.

3.4 Technology choices

When choosing the technology to use in our Application, we decided to use a range of new and innovative technologies ranging from IDEs to server clients that are in different stages of development ranging from Pre-Alpha such as MongoDB Realm Integration to a stable working environment with the Kotlin Language and Android.

3.4.1 Kotlin

As a team, we wanted to learn a new programming language as a challenge for our project and to increase our skill set as Software Developers. One of our lecturers during one of our online seminars from our module Applied Project Minor Dissertation mentioned the Kotlin language. This led us to research this language and understand what it could be used for. We saw that Kotlin was an up-and-coming language that was primarily used for mobile Applications in Android as of September 2020 with hopes of iOS support being implemented in the near future.

3.4.2 IntelliJ IDEA

At the start of the college year, we were introduced to IntelliJ in our Distributed Systems module. We soon realised that JetBrains who are the founders of IntelliJ are also the designers of the Kotlin programming language. With this information, we decided to use this as our chosen development environment.We soon realised that we had the wrong version of IntelliJ installed on our devices. We needed to have the IntelliJ IDEA Ultimate to be able to access a Mongo Database. We got this free with the GitHub Student Developer Pack. This was an issue that was unforeseen and time-consuming but easily rectified. With JetBrains owning both Kotlin and IntelliJ writing the code was seamless and allowed testing to constantly occur. IntelliJ allowed the use of an Android mobile phone emulator to run these tests.

3.4.3 MongoDB

MongoDB is a NoSQL database service that allows a user to store data from an application for future use. The data stored can be queried in searches that outputs specific results. These results can be processed by the application and read out to the user. We chose to use MongoDB as our server because of our previous experience of developing applications with it as the back-end storage.

To utilize a MongoDB database. We had to create a MongoDB Atlas cluster to be used with a MongoDB Realm application. This is where the user data can be read into the Kotlin application. This can also be used to send data to the cluster through the created Realm application for storage. Realm reads in a cluster that it has permission to access through a partition key and sends data to the Kotlin application.

3.4.4 JustInMind

In order to conceptualize and visualise our application's front-end layout, we had to begin with prototyping our design. While researching design tools for mobile application development, we came across a prototyping tool called JustInMind.

It is a high-level prototyping tool used to create high-quality wireframes that allow developers to visualise their product in real-time before finalizing the design of the mobile application. It assisted us in figuring out how we want the user to navigate the application.

We tried to design the app with different orientations which can be seen in Figure 3.3 which shows the prototype for the recommended days page within the application.

In Figure 3.2 it displays the simplicity of designing features within the page on the App, this image shows how designing a prototype for a drop-down menu can be easy to design and how the design feels in correlation to the design of the page.

The Image as seen in Figure 3.1 shows the design of the prototype for the login/Sign Up page for the user, it gives a simple understanding of the basic functionality required on this page within the Application.

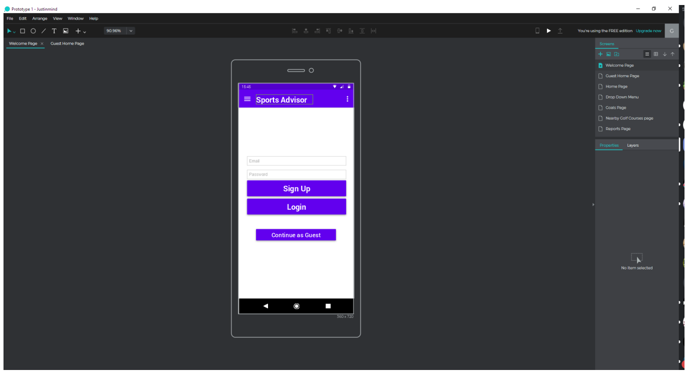


Figure 3.2: Sign up prototype within JustInMind

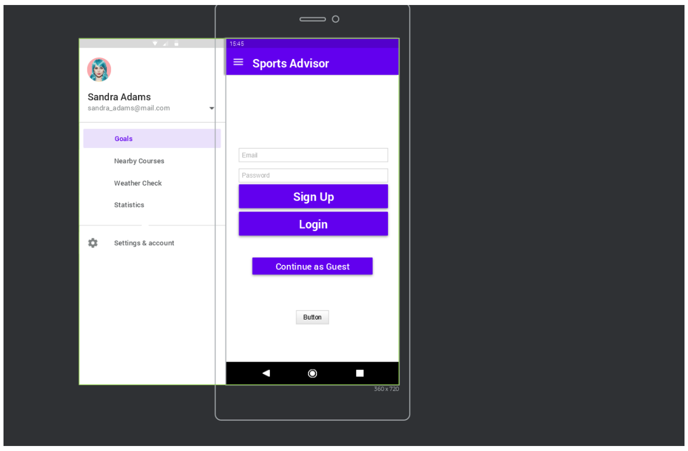


Figure 3.3: Drop Down Menu prototype within JustInMind

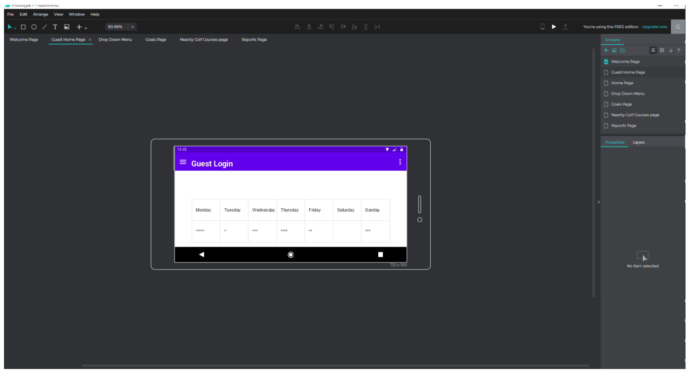
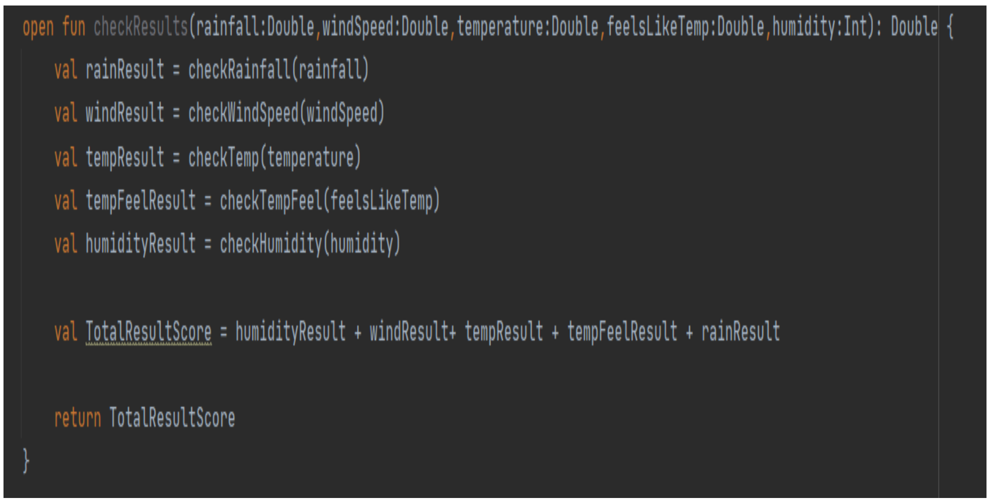


Figure 3.4: Best Days to Play prototype within JustInMind

3.5 Algorithm

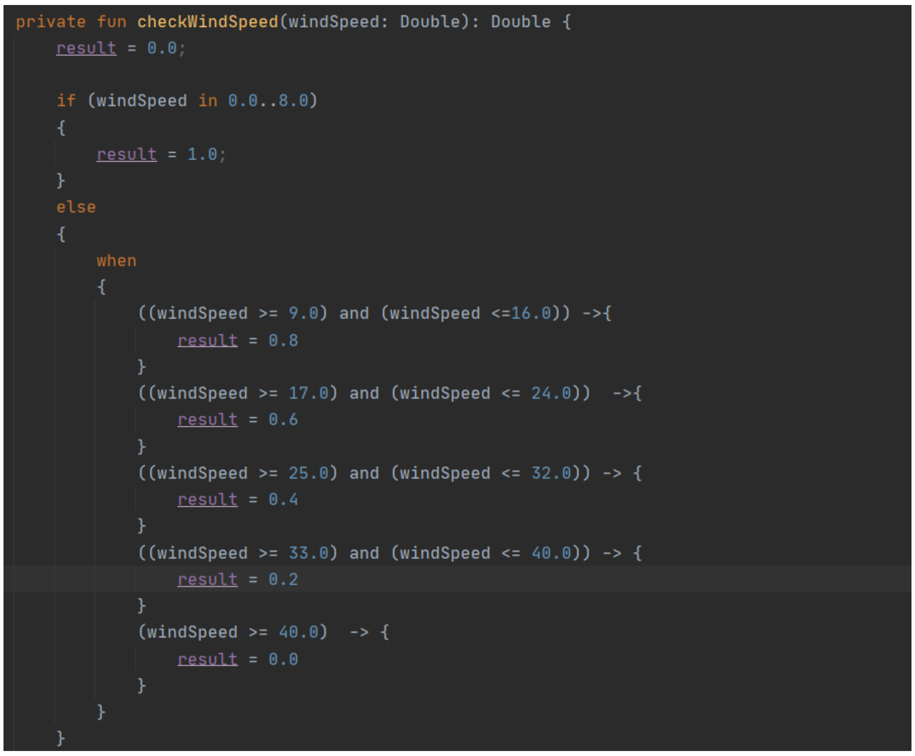
The algorithm we have implemented for the app is used for giving the user feedback on the weather conditions for playing golf during a specific hour on the current date. We generate JSON data from an API call and it contains an abundance of different weather variables which we have saved into variables that can be accessed in the application. We have chosen 8 variables from the list that are key to what hour is best suited to playing a round of golf. They are sunrise, sunset, hour, amount of rainfall in millimetres, temperature, wind speed, and humidity. All these variables are given back to the user alongside a star rating. The star rating is out of 5, with 0 being the worst and 5 being the best.



```
open fun checkResults(rainfall:Double,windSpeed:Double,temperature:Double,feelsLikeTemp:Double,humidity:Int): Double {
    val rainResult = checkRainfall(rainfall)
    val windResult = checkWindSpeed(windSpeed)
    val tempResult = checkTemp(temperature)
    val tempFeelResult = checkTempFeel(feelsLikeTemp)
    val humidityResult = checkHumidity(humidity)

    val TotalResultScore = humidityResult + windResult+ tempResult + tempFeelResult + rainResult

    return TotalResultScore
}
```

Figure 3.5: Star rating function

We based the ratings generated inside the userResults.kt class from research we gathered and personal experience on what the optimal conditions for enjoying golf would bed. [11] In the below image we illustrate how we generate a star rating for a variable, every variable is worth 1 star each but that star is broken down into a result between 0 and 1. The result is then used to generate a total score in the check Results function.



```
private fun checkWindSpeed(windSpeed: Double): Double {
    result = 0.0;

    if (windSpeed in 0.0..8.0)
    {
        result = 1.0;
    }
    else
    {
        when
        {
            ((windSpeed >= 9.0) and (windSpeed <=16.0)) ->{
                result = 0.8
            }
            ((windSpeed >= 17.0) and (windSpeed <= 24.0))  ->{
                result = 0.6
            }
            ((windSpeed >= 25.0) and (windSpeed <= 32.0)) -> {
                result = 0.4
            }
            ((windSpeed >= 33.0) and (windSpeed <= 40.0)) -> {
                result = 0.2
            }
            (windSpeed >= 40.0)  -> {
                result = 0.0
            }
        }
    }
```

Figure 3.6: Wind Rating function

Image to show output from rating

Chapter 4

Technology Review

4.1 Kotlin

We found Kotlin to be less verbose, easier to read as a programming language compared to such languages as Java or C. It is a high-level language that can be used to produce mobile applications. Its coding style is similar to python but with more features and ways to implement it.

The Kotlin language is a very good interpreter of other languages which helps when trying to write code without having a great grasp of the language. This is very important when you are beginning to learn the language or when you cannot find a way to write a method in Kotlin that you have done previously in a different language such as Java.

Despite both Kotlin and Java being native languages for writing Android applications, Kotlin is seen by many as a cleaner and concise way to write code for mobile than Java[21][22]. It can reduce the number of lines of code that need to be written for the application - this comparison can be seen in the code Excerpts below.

Below can be seen two examples of code one written in Kotlin and the other in Java, both pieces of code perform the same function but are equally different. As we can see in the Kotlin Code excerpt, the code tends to be more human-readable and easier to understand for people less acquainted with either of the languages.

Kotlin Code Excerpt

```
@Throws(IOException::class)
fun run(url: String?): String? {
    val request: Request = Request.Builder()
        .url(url.toString())
        .build()
    client.newCall(request).execute().use {
    response -> return response.body!!.string()} }
```

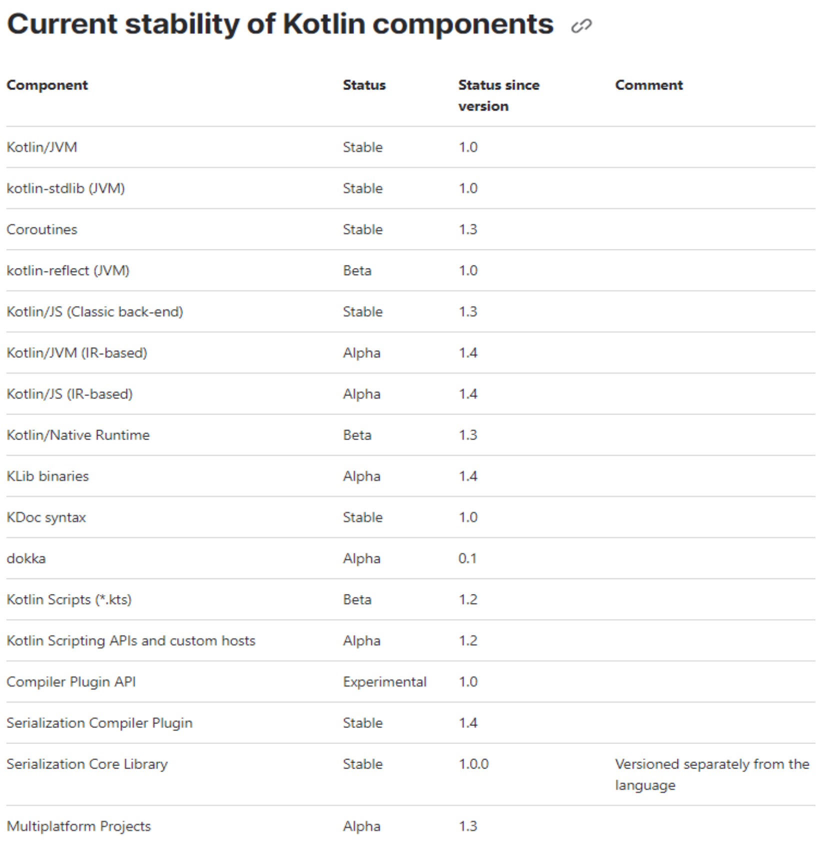
Java Code Excerpt

```
OkHttpClient client = new OkHttpClient();
String run(String url) throws IOException {
    Request request = new Request.Builder()
        .url(url)
        .build();
    try (Response response = client.newCall(request).execute()){
     return response.body().string(); }}
```

The Above code is taken from [1] and is adapted for Kotlin using [20]

Limitations of the Kotlin Language

Kotlin comes with many amazing features that assist the Developer in writing efficient and easy-to-understand projects and Applications but with all of these innovative and useful tools that come with the Kotlin language, it is always changing. As the Kotlin programming language is relatively new in comparison to other programming languages such as Java and C++, the language itself and its components vary in stages within a Software release life cycle, an example of which would be the Kotlin/JVM is in stable development since version 1.0 but the component for Multiplatform projects is still in Alpha version 1.3 as of the May 2021 which can be seen in Figure 4.1.[2]



Current stability of Kotlin components

Component	Status	Status since version	Comment
Kotlin/JVM	Stable	1.0	
kotlin-stdlib (JVM)	Stable	1.0	
Coroutines	Stable	1.3	
kotlin-reflect (JVM)	Beta	1.0	
Kotlin/JS (Classic back-end)	Stable	1.3	
Kotlin/JVM (IR-based)	Alpha	1.4	
Kotlin/JS (IR-based)	Alpha	1.4	
Kotlin/Native Runtime	Beta	1.3	
KLib binaries	Alpha	1.4	
KDoc syntax	Stable	1.0	
dokka	Alpha	0.1	
Kotlin Scripts (*.kts)	Beta	1.2	
Kotlin Scripting APIs and custom hosts	Alpha	1.2	
Compiler Plugin API	Experimental	1.0	
Serialization Compiler Plugin	Stable	1.4	
Serialization Core Library	Stable	1.0.0	Versioned separately from the language
Multiplatform Projects	Alpha	1.3	

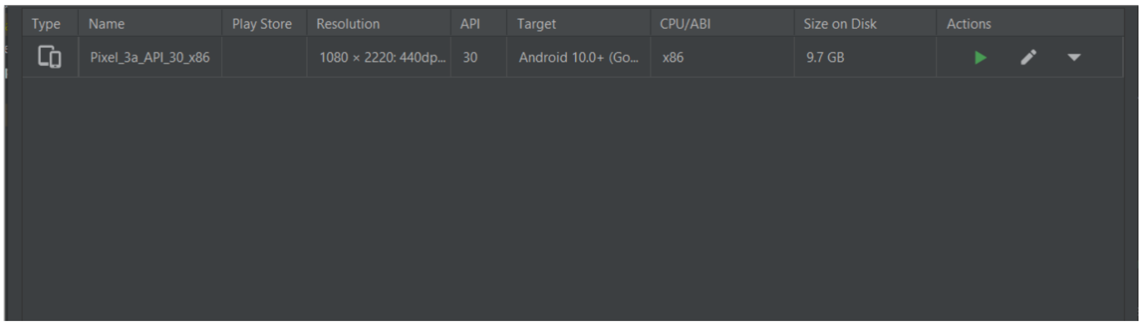
Figure 4.1: Stability of Kotlin and its components

Due to the Multiplatform component not being in a stable release version, we decided to solely focus on development for Android devices as this is included with the stable release of Kotlin/JVM and along with our experience in creating applications for Android devices during our time at GMIT.

We initially created our base Application with the intention of utilising the multiplatform component for our application. We soon found that it was very difficult to run and in some instances would not compile and would sometimes render the Application completely inoperable. This came as a cause for concern for us in the early stages of development and initially delayed our development by two weeks. It made us reflect on the feasibility of the component and if it would cause major issues later on in the development life cycle of the Application and as a result was ultimately removed from the Application due to being too unstable, with a new project entirely designed for the Android platform being created instead.

4.2 IntelliJ IDEA's

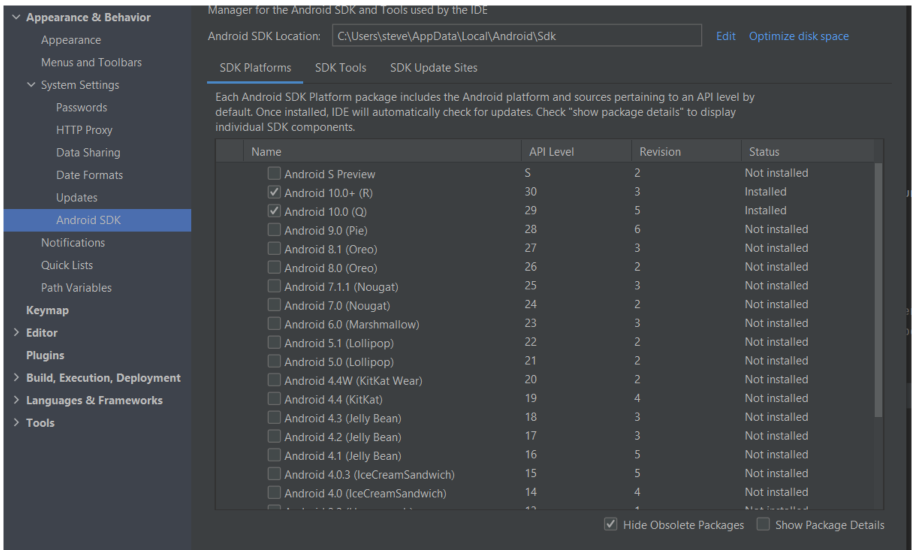
The development environment provided by IntelliJ allows for the development of several different types of programs including Mobile Applications. The IDE contains a built-in android development environment with Kotlin being the default language.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel_3a_API_30_x86		1080 × 2220: 440dp...	30	Android 10.0+ (Go...	x86	9.7 GB	

Figure 4.2: Android Device Kit

The image above shows the emulated device customisation menu. In here you can add a virtual device to run the android application on. The ability to modify what Android SDK used within the IDE was very important for testing the project. That feature is a great way to debug a mobile application efficiently and in real-time.



Manager for the Android SDK and Tools used by the IDE

Android SDK Location: C:\Users\steve\AppData\Local\Android\Sdk Edit Optimize disk space

SDK Platforms SDK Tools SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, IDE will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
☐ Android S Preview	S	2	Not installed
✓ Android 10.0+ (R)	30	3	Installed
✓ Android 10.0 (Q)	29	5	Installed
☐ Android 9.0 (Pie)	28	6	Not installed
☐ Android 8.1 (Oreo)	27	3	Not installed
☐ Android 8.0 (Oreo)	26	2	Not installed
☐ Android 7.1.1 (Nougat)	25	3	Not installed
☐ Android 7.0 (Nougat)	24	2	Not installed
☐ Android 6.0 (Marshmallow)	23	3	Not installed
☐ Android 5.1 (Lollipop)	22	2	Not installed
☐ Android 5.0 (Lollipop)	21	2	Not installed
☐ Android 4.4W (KitKat Wear)	20	2	Not installed
☐ Android 4.4 (KitKat)	19	4	Not installed
☐ Android 4.3 (Jelly Bean)	18	3	Not installed
☐ Android 4.2 (Jelly Bean)	17	3	Not installed
☐ Android 4.1 (Jelly Bean)	16	5	Not installed
☐ Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
☐ Android 4.0 (IceCreamSandwich)	14	4	Not installed

✓ Hide Obsolete Packages ☐ Show Package Details

Figure 4.3: Android Software Development Kit

The SDK manager on IntelliJ allowed us as developers to run the application on different versions of the Android system. When you choose an SDK, it gives you a percentage of how many people who use Android will be able to use your device. The more up-to-date version - meant fewer people could run the app, we when with the Android Oreo version for our virtual phone which is Android 10.0 but we set our minimum SDK version to 28. This allowed the phone to run on Android 9.0(Pie) as well as the version we set up on the virtual device.

4.3 API's Tested

4.3.1 Open Weather API

Open Weather is an API that allows the user to retrieve weather data from any location using its NWP (Numerical Weather Prediction) Model. This API has been designed to be used universally in any Language to gather data. It gathers this data using a proprietary convolutional neural network that collects and processes a wide range of data sources to cover any location and consider the local nuances of climate. (Quote from https://openweathermap.org/guide - Openweather NWP model)

4.3.2 Geocoding API

In order to retrieve data with the Open Weather API, we needed the longitude and latitude of a chosen golf course selected by the user. With the help of the Geocoding API, it gives the ability to convert street name data into longitude and latitude values for the Open Weather API to search weather data based on that location.

4.3.3 AccuWeather API

After a lot of testing and attempting to retrieve data using the Open Weather API and Geocoding API, we found it was very difficult to retrieve, store and convert the data from XML that was provided by the Open Weather API. We then came across the Accuweather API that searches for locations based off of their own location codes which we then use to offer the user a list of golf course locations within Co. Galway that they could choose from instead of having to search for courses themselves.

By offering this list we can accurately show the conditions for these particular golf courses of choice instead of being able to search for weather conditions for cities and towns instead of the actual golf courses.

The data provided from the API is then able to be processed by our Algorithm to provide a rating for the conditions based on two search features, a 12-hour search for that particular day or a search based on the next 5 rolling days, the 12-hour search will provide the rating based on the next twelve hours and will provide a rating for those hours, and then the current conditions search will give a general rating for the current hour when accessed by the guest functionality. Due to all these features we decided to use this API to help us retrieve the weather data to be used to generate ratings.

4.4 MongoDB

To utilise the MongoDB database, we had to use two of its three main core features. The two features we use are Atlas and Realm. Atlas and Realm can be integrated in a way that allows for a seamless connection to and from the Android Application. The MongoDB Atlas uses a server that can be either free or paid for use by a set charge per hour. We choose a free tier server based in N.Virginia (us-east-1) run by Amazon Web Service(AWS). This server was chosen because the Irish server does not work with MongoDB Realm as the server for Ireland is only using MongoDB version is 4.2 but Realm runs on MongoDB version 4.4 and above.

4.4.1 Cluster

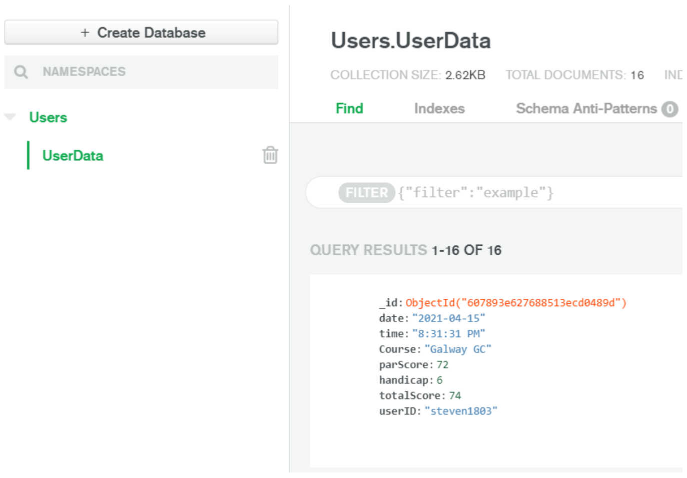
It is the word associated with having several MongoDB servers working as a connected system. MongoDB distributes data in 2 ways: replica set and sharded clusters. The main objectives of a MongoDB cluster are to be able to read and write several nodes. The data is separated into the different nodes of the fragment. For our database we are using the replica set due to its ability to be used multiple times throughout an application.[4]

1. **Replica Set:**
   This is when the data is sent across several servers without the data changing. This is a layer of protection from a server failure.

2. **Sharded Cluster:**
   This is where the data is fragmented into parts of the data to be carried by a different server. This is done to have larger datasets and have better performance.

4.4.2 Atlas

Atlas is the area in which data is stored within the MongoDB system. it allows its users to create a database for a specific user that will only store their data in a collection.
This is great for protecting user data. The data can only be accessed by inputting a user id. This user id is unique and also acts as the user's username that will be always shown in the application. With the data for a user protected within the MongoDB database, this allows the application to output data to the user that is only relevant to them. That feature was a major reason that to chose to use MongoDB in our application.
The user can store the name of the golf course they played at and the score they got that day. It will also save the par for the course that date and the overall total score for the round of golf.



```
{"filter":"example"}

QUERY RESULTS 1-16 OF 16

_id: ObjectId("607893e627688513ecd0489d")
date: "2021-04-15"
time: "8:31:31 PM"
Course: "Galway GC"
parScore: 72
handicap: 6
totalScore: 74
userID: "steven1803"
```

Figure 4.4: MongoDB database with a User

The highlighted field in the above figure is the authentication key that will need to be inputted by the user to access the database associated with them. Without this key being correctly inputted, the user will not get any data from the MongoDB cluster. This is how MongoDB protects user data.

4.4.3 Realm

The Realm feature of MongoDB is how MongoDB now interprets a Mobile Application. It creates an application that can be used to access a cluster - The cluster is what Atlas creates to store the user data.
The App will have a unique ID that is called in the Mobile Application back-end to link your application to MongoDB. It creates a user that is linked to a specific database in the cluster. This user can only be authenticated by entering the correct email and password. The app can and does have rules that limit what data a user can access to modify or view.

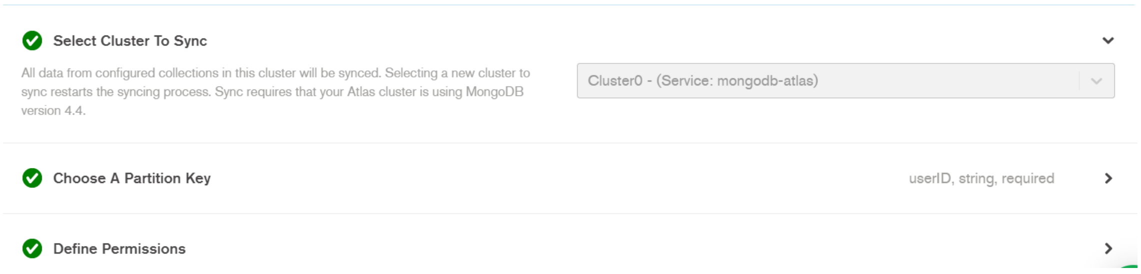
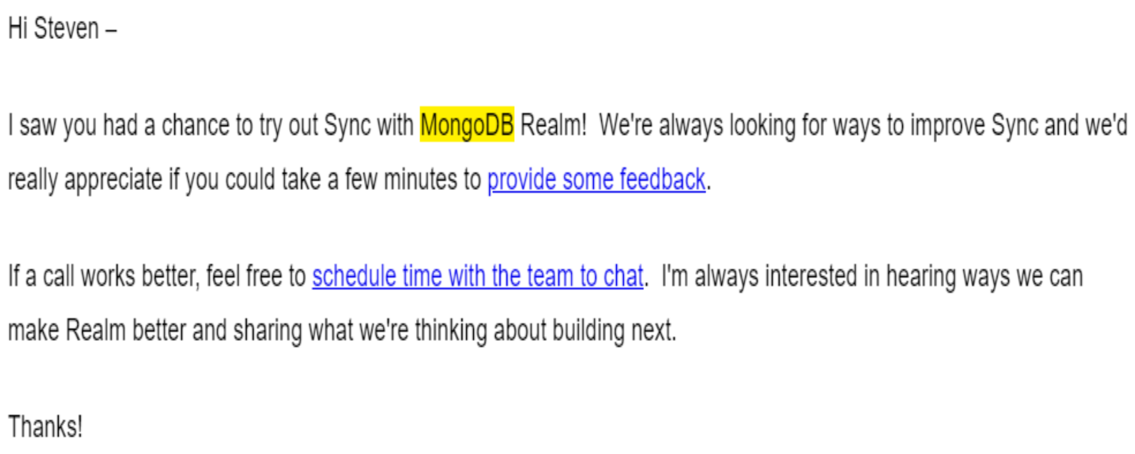

Select Cluster To Sync
All data from configured collections in this cluster will be synced. Selecting a new cluster to sync restarts the syncing process. Sync requires that your Atlas cluster is using MongoDB version 4.4.
Cluster0 - (Service: mongodb-atlas)
Choose A Partition Key
userID, string, required
Define Permissions

Figure 4.5: MongoDB Sync configuration

The above figure is an image of the MongoDB sync configuration. This is where the developer connects to the Atlas cluster by implementing a partition key. This key is how a cluster is made available to the user. The permissions for what a user can do with the database it has connected to are set here as well. We have allowed the user to both read and write to the collection but the data is protected by the userID key created in the collection itself.
For this project, Realm functioned as the middle man between the application and the MongoDB Server. It allowed the data to be sorted in a manner in which is easy to understand and utilize. Despite the Realm Sync still being in a developmental phase, it is of a high enough standard to work with no connectivity issues on their end.
We have been contacted by the MongoDB Realm team to give feedback to help them see what issues have occurred and if it has worked well for us.


Hi Steven –

I saw you had a chance to try out Sync with MongoDB Realm! We're always looking for ways to improve Sync and we'd really appreciate if you could take a few minutes to provide some feedback.

If a call works better, feel free to schedule time with the team to chat. I'm always interested in hearing ways we can make Realm better and sharing what we're thinking about building next.

Thanks!

Figure 4.6: MongoDB Realm email for feedback with names omitted

Realm allows the creation of App users. This was the best way we found to make data private and protected. The user will have to log into the server with their email, and password. These are created in the app and are stored on the Realm App external storage system. The user needs to log in with an email and password to store any round of golf score that they input into the ScoreActivity.kt class. This can be seen afterwards inside YserHistory.kt class.

4.5 GitHub

GitHub is the version control platform we used for our project which allows us to track and manage our source code with version control in a collaborative fashion. We used a private repository which allowed us to control who could access our code as well as who could contribute to our code. It allowed us to use branches to merge our code so we can work on separate features and not worry about breaking each other's code.

Since it is directly connected to git, we were able to push and pull code directly from the terminal or the GitHub Desktop application which we had experience in, and as a result, was our preference over using IntelliJ's inbuilt GitHub features. The GitHub website has a solid user interface, the service is stable and reliable so we had no worries about hosting our code on their site. GitHub also offers a student pack which gave the team free access to developer tools, that we incorporated into our project including MongoDB Realm and IntelliJ Ultimate.

4.6 Communication Application's

4.6.1 Discord

Discord is a free instant messaging and Voice over Internet Protocol service which is generally targeted for gaming or other online communities to communicate together on servers. Our group was already well acquainted with its features and was comfortable with setting up our own server for the group's project. Discord is very similar if not identical to the untrained eye to the business communication platform Slack which is used by businesses in a workplace environment the main difference being that Discord is completely free whereas Companies pay a fee to use Slack.

Discord then served as an appropriate alternative to Slack as it offered the same features including persistent chat rooms organized by topic, private groups, direct messaging, screen sharing, and online voice meeting capabilities.

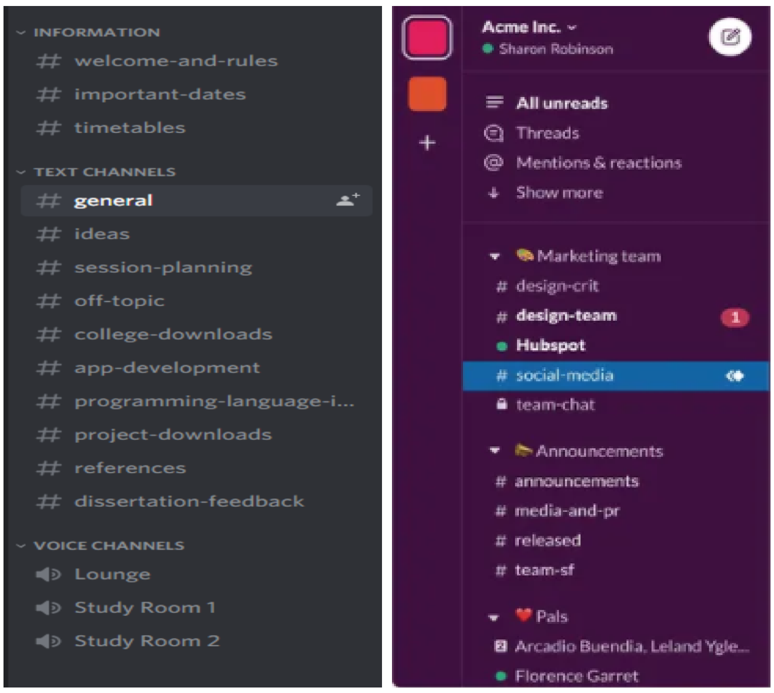


Figure 4.7: Similarities between Discord and Slack

4.6.2 Teams

Microsoft Teams is a collaborative workspace that is fully integrated with Microsoft Office 365 Services. Teams allows for scheduled voice and screen sharing meetings to be arranged via a calendar layout so that users can organise multiple meetings throughout a day, as well as schedule meetings to automatically occur on a daily, weekly or monthly basis such as in our case we were able to set up our meeting with our supervisor to automatically schedule itself every Monday at 11:30.

We also found Teams to be a lot more user-friendly to users who are not as technologically skilled, rarely does Teams encounter voice chat or microphone connectivity issues.

It has both a desktop application and a browser application formats for users on computers which allows flexibility for a user who might be borrowing a machine or have low storage they can use the web app version through a browser.

Similarly, users can use an Android or iOS app for when the situation is more suitable for example replying to a direct message or meeting invitation whilst away from their desktop or laptop device. Its also worth noting that if a team is using an Office platform such as Excel, PowerPoint, or Word, the members of the team can work on the same document at the same time which is a major advantage for people using Office platform tools. The only downside of teams which is a common criticism of Microsoft software would be its UX can be frustrating, while it has improved in recent updates the UX is nowhere near as smooth as Slack or Discords in terms of navigating teams or groups you are a part of and its message feeds.

4.7 Dissertation Proof-Reading Tools

When we began writing up this document, we wanted to use a tool to check the grammar used, give a guide to what changes need to be made and overall make this document more easier to read. We realised this when we started to give our supervisor the document to get constructive feedback on what we can improve on. To combat this we decided to use Grammarly.

Grammarly is one of the most popular tools used for proof reading files. It allows the user to upload a file of any size and it is proof-read within a minute of uploading the documents. The only issue we had with this tool is that it would give hints to change the way words are written into the American variations of English words such as colour or realised. It wanted us to change those 2 words into color and realized. The hints it gave helped us convert sentences into words that better convey what we wanted to say.

With the use of Grammarly we were able to fix any grammatical errors of the document within half an hour and without having to read the document repeatedly. It allowed us to work efficiently and manage our time better. Grammarly is able to be added to a browser with an extension or as a desktop app on windows. The ability to use it in many locations on a computer device made the use of it more convenient.
Here is a link to the Grammarly website: `https://app.grammarly.com`

Chapter 5

System Design

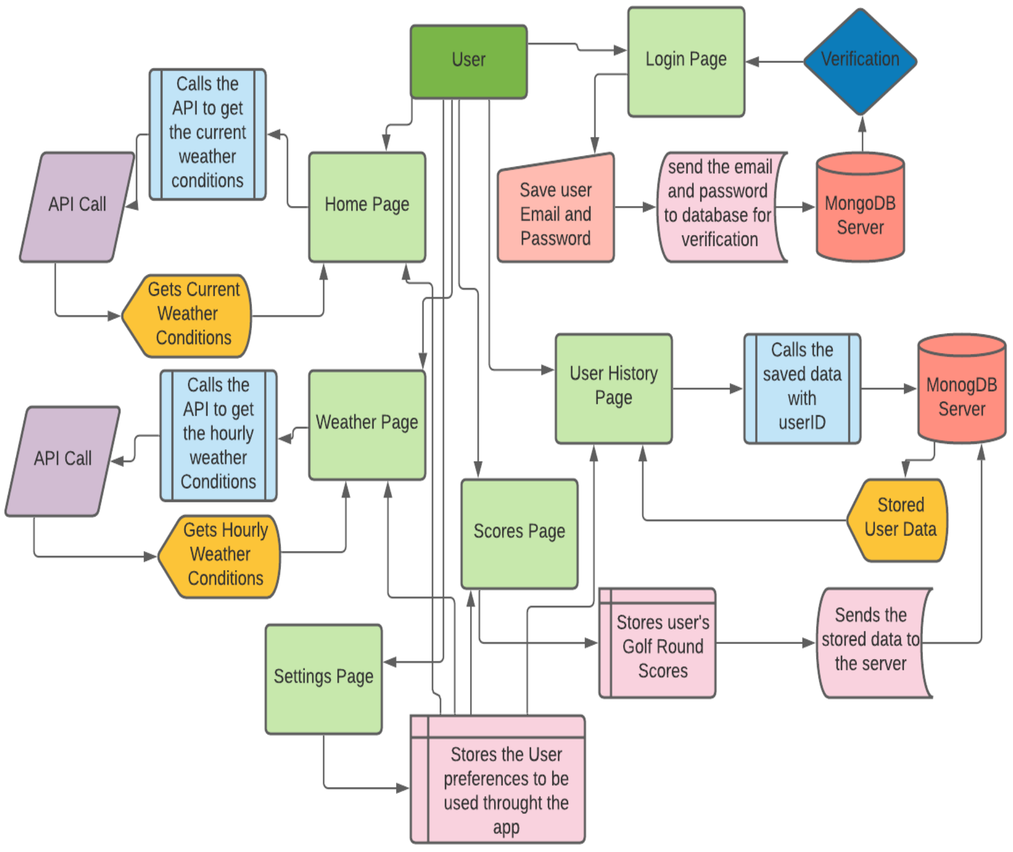

User
Login Page
Verification
Calls the API to get the current weather conditions
API Call
Home Page
Save user Email and Password
send the email and password to database for verification
MongoDB Server
Gets Current Weather Conditions
User History Page
Calls the saved data with userID
MonogDB Server
Calls the API to get the hourly weather Conditions
API Call
Weather Page
Stored User Data
Gets Hourly Weather Conditions
Scores Page
Stores user's Golf Round Scores
Sends the stored data to the server
Settings Page
Stores the User preferences to be used throught the app

Figure 5.1: Data Flow Diagram of the Application

With our system, we have three main layers that integrate with one another to build our functional application. The front-end part of the application deals with the visuals and functionality that will be delivered to the user. It communicates with the back-end of the application which stores and retrieves data for the user upon interacting with the front-end elements. The Data Retrieval tools are used in collaboration with the front-end to retrieve and present the current weather data for now and the next twelve hours as a set of ratings for the optimal playing conditions for each hour.

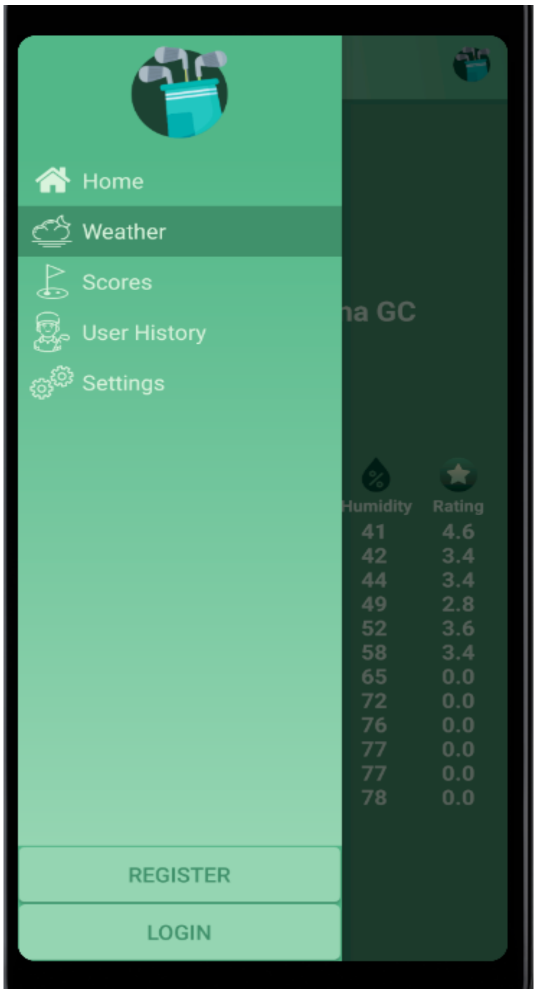
5.1 Front End

For the front-end of our application we set out to make a user interface that would be both easy to navigate and appealing for the user to look at. We wanted the user to able to get to any page from the current page they were in with as few commands as possible as well. Thus it was decided that a drawer with each page was appropriate for our design. With a drawer the user can access an array of options in this case page titles by swiping left or tapping the three bars in the top left of the screen. From here they can select the page they want and have that page displayed in front of them.

5.1.1 Android

Navigation

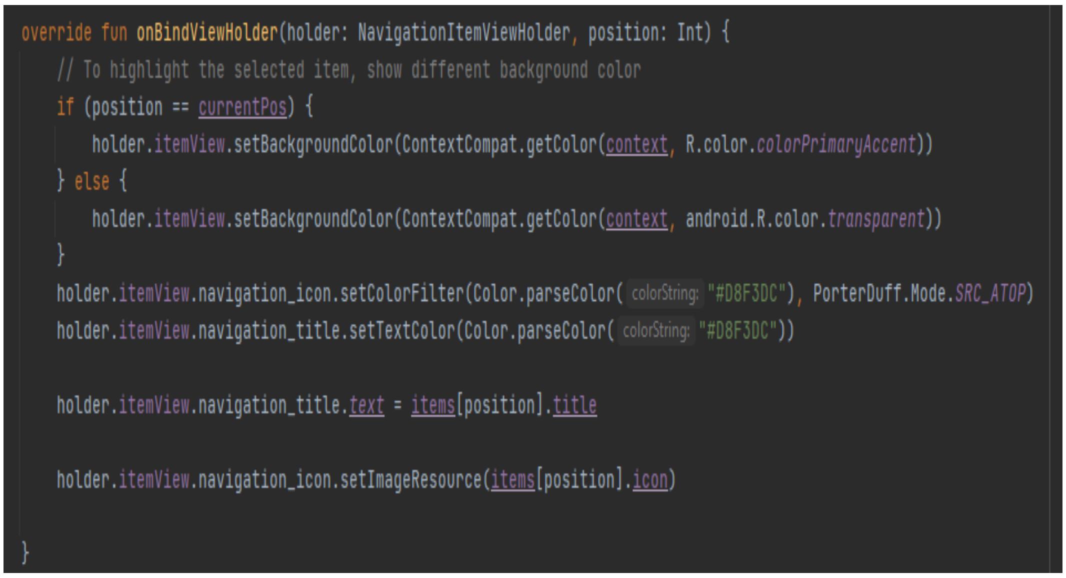
When the app first launches on the Main Activity, in the onCreate method the apps drawer layout, toolbar and RecyclerView are setup. The Recycler-TouchListener.kt class will detect which row of the RecyclerView is selected. Below the RecylcerView is the two buttons that will take you to the login and register pages.



Humidity	Rating
41	4.6
42	3.4
44	3.4
49	2.8
52	3.6
58	3.4
65	0.0
72	0.0
76	0.0
77	0.0
77	0.0
78	0.0

Figure 5.2: Drawer Layout displaying menu items in a RecyclerView

NavigationRVAdapter.kt takes an array list of menu items and the current position selected, when the user selects a menu option, the listener will execute the code for each position, this will load in a fragment or a new activity and then the updateAdapter will be called to update all the menu items appearance to reflect this interaction.

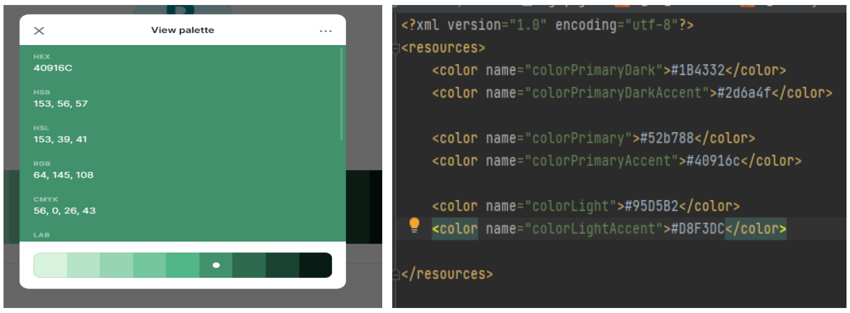


```
override fun onBindViewHolder(holder: NavigationItemViewHolder, position: Int) {
    // To highlight the selected item, show different background color
    if (position == currentPos) {
        holder.itemView.setBackgroundColor(ContextCompat.getColor(context, R.color.colorPrimaryAccent))
    } else {
        holder.itemView.setBackgroundColor(ContextCompat.getColor(context, android.R.color.transparent))
    }
    holder.itemView.navigation_icon.setColorFilter(Color.parseColor( colorString: "#D8F3DC"), PorterDuff.Mode.SRC_ATOP)
    holder.itemView.navigation_title.setTextColor(Color.parseColor( colorString: "#D8F3DC"))

    holder.itemView.navigation_title.text = items[position].title

    holder.itemView.navigation_icon.setImageResource(items[position].icon)

}
```

Figure 5.3: onBindViewHolder adjusts menu items appearance

XML

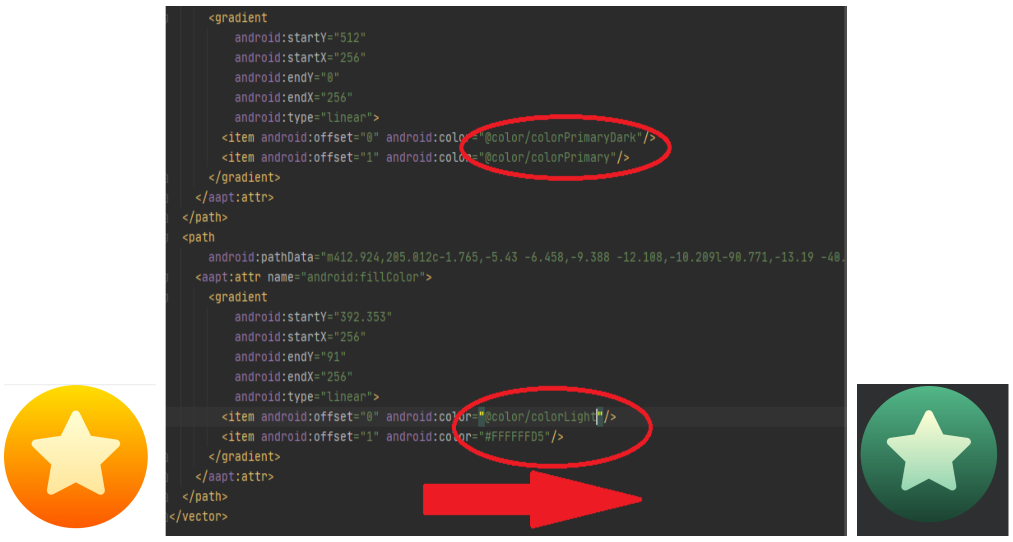
The XML section of the project contains the three XML folders which were the most used being drawable, layout and values. Drawable contains the xml data for all the icons and themes for the app's background and buttons. We used a colour scheme which we got from coolors.co. [12] This site was a great tool, as we could enter in a colour for our app in this case green and the site will generate a colour scheme to compliment that colour. We could get the hexadecimal value for each of the colours in the scheme and add them to a color.xml file that we could give a descriptive name for each colour and apply these throughout the front-end and what resulted was a clean and consistent look for the end user on every page.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimaryDark">#1B4332</color>
    <color name="colorPrimaryDarkAccent">#2d6a4f</color>

    <color name="colorPrimary">#52b788</color>
    <color name="colorPrimaryAccent">#40916c</color>

    <color name="colorLight">#95D5B2</color>
    <color name="colorLightAccent">#D8F3DC</color>

</resources>
```

Figure 5.4: colour scheme generated by coolors.co(Left), shown in XML(Right)

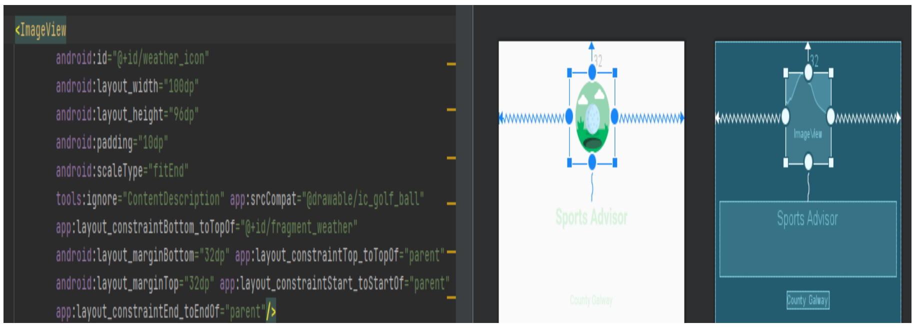
The XML colours were used to convert button design and icons we sourced from the web to fit the colour scheme of the app. We could download SVG icons and convert them into an XML vector assest in Android Studio, then apply our colour scheme. [13]



```
        <gradient
            android:startY="512"
            android:startX="256"
            android:endY="0"
            android:endX="256"
            android:type="linear">
          <item android:offset="0" android:color="@color/colorPrimaryDark"/>
          <item android:offset="1" android:color="@color/colorPrimary"/>
        </gradient>
      </aapt:attr>
    </path>
    <path
        android:pathData="m412.924,205.012c-1.765,-5.43 -6.458,-9.388 -12.108,-10.209l-90.771,-13.19 -40.
      <aapt:attr name="android:fillColor">
        <gradient
            android:startY="392.353"
            android:startX="256"
            android:endY="91"
            android:endX="256"
            android:type="linear">
          <item android:offset="0" android:color="@color/colorLight"/>
          <item android:offset="1" android:color="#FFFFFD5"/>
        </gradient>
      </aapt:attr>
    </path>
</vector>
```

Figure 5.5: Original Star icon(Left),Converted to fit app theme via xml color values, Result(Right)

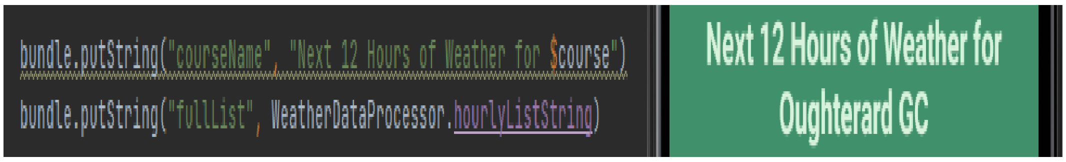
The layout folder contained the XML for each pages layout which defines the structure for a user interface in the app. Inside the defined layout we could dictate where on the page to display views and interactables such as textView, buttons and imageView. The majority of the layouts in the app use the constraint layout which allowed us to attach view components either vertically or horizontally, which suited our design as we had most components constrained to the top or bottom of one another.



```
<ImageView
        android:id="@+id/weather_icon"
        android:layout_width="100dp"
        android:layout_height="96dp"
        android:padding="10dp"
        android:scaleType="fitEnd"
        tools:ignore="ContentDescription" app:srcCompat="@drawable/ic_golf_ball"
        app:layout_constraintBottom_toTopOf="@+id/fragment_weather"
        android:layout_marginBottom="32dp" app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="32dp" app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>
```

Figure 5.6: ImageView constrained to the top, sides and to the TextView below

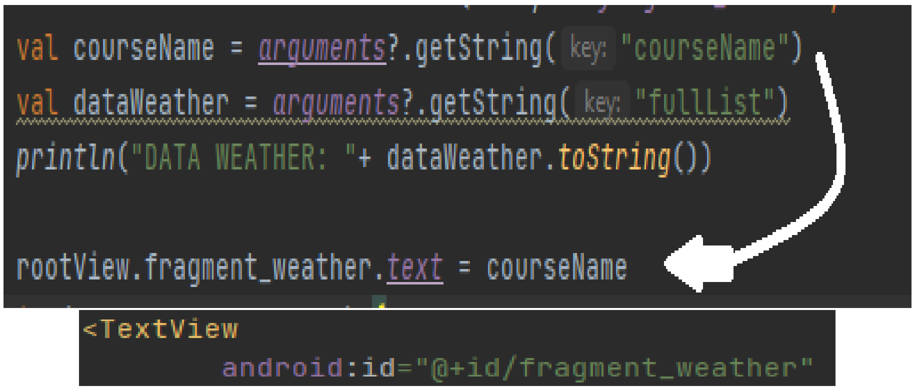
Each View contains an id which allows the XML to be altered by the fragment or activity it's layout is contained within. For Example in the weather_fragment.xml page the TextView for course name is assigned whatever the arguments are passed to the WeatherFragment.kt page's courseName variable.



```
bundle.putString("courseName", "Next 12 Hours of Weather for $course")
bundle.putString("fullList", WeatherDataProcessor.hourlyListString)
```

Next 12 Hours of Weather for Oughterard GC

Figure 5.7: Main Activity puts string into bundle with key



```
val courseName = arguments?.getString( key: "courseName")
val dataWeather = arguments?.getString( key: "fullList")
println("DATA WEATHER: "+ dataWeather.toString())

rootView.fragment_weather.text = courseName
```

```
<TextView
        android:id="@+id/fragment_weather"
```

Figure 5.8: arguments inside Fragment change the .text variables value for TextView with corresponding id

Activities

The Application contains five Activities including one MainActivity which hosts the Fragments of the application.The Main Activity changes its display by hosting Fragments within its layout. It hosts the Home, Weather and Settings fragments. Any data that the Fragments use is passed in a Bundle from the MainActivity where it calls upon the different elements of the back-end.

The Login and Register Activities contain XML drawables for their page icons as well as backgrounds which blend between two different colour scheme values. They implement a cardview which places the input Boxes(2) above the background box(1) and the Login/Register Buttons(3) appear on top of the input Boxes giving the page a 3D impression.

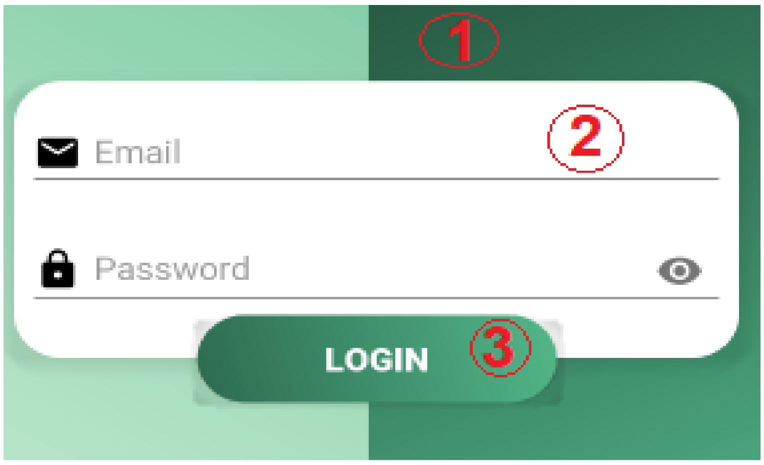
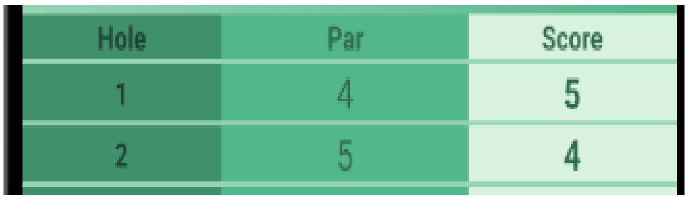


Figure 5.9: Layered display for 3D effect

The ScoreActivity page is held within a simple constraint layout which contains all of its inputs in a table layout. when the player has entered in all the input fields then they can press the save score button.When the button is pressed the Activity will calculate the players scores and update the Score fields at the bottom of the Scorecard. The Scorecard uses contrasting colours from the scheme to differentiate between the different columns with each colour representing a different field either Hole,Par or Score.



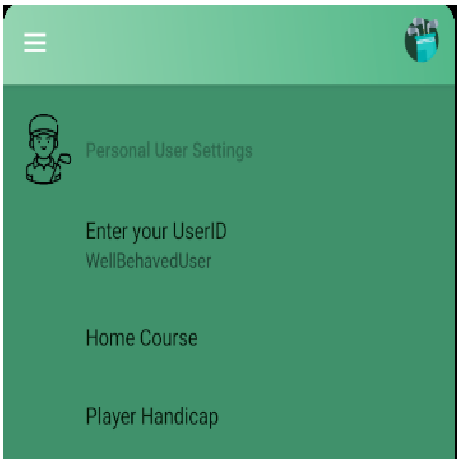
Hole	Par	Score
1	4	5
2	5	4

Figure 5.10: Contrasting Columns for Scorecard

Fragments

The Fragments Settings, Home and Weather are all portions of UI which live within the MainActivity. The Home and Weather Fragments display the result of the back-end getting weather data and processing it to get the results for the user. For these Fragments to get data from MainActivity, they are passed a "Bundle" object into its arguments which store's data types with a key. The Fragments layout will replace the current layout within the Main-Activity rootView. They then can then access the data they want via the key for the bundle and can use it to change XML data inside its XML layout which allows us to update to relevant weather data.

The Settings fragment displays a PreferenceScreen which allows the user to change the course they wish to see weather data for, their handicap for calculating score and their user name. By setting these variables as preferences, The app will locally save the values set so the user will always have the same user name, handicap and the last course they had set will be their default. We can easily call them in any activity or fragment for use in the front or back-end of the application with a key which made for rapid and simple implementation.



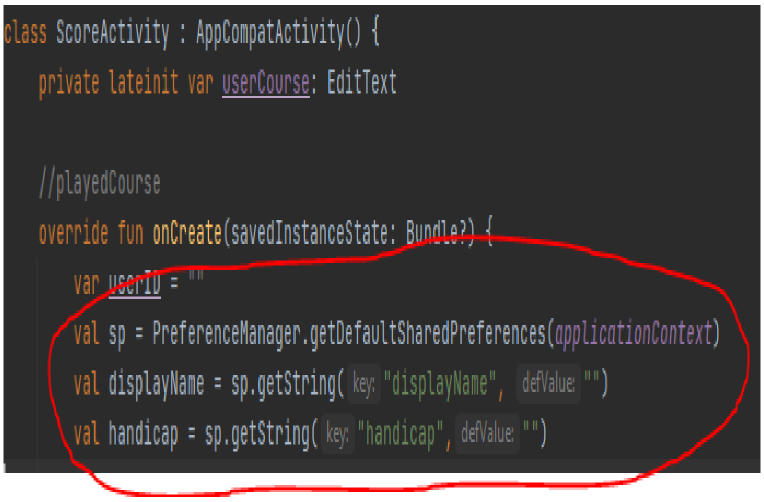
Personal User Settings

Enter your UserID
WellBehavedUser

Home Course

Player Handicap

Figure 5.11: User can set their preferences in settings fragment



```
class ScoreActivity : AppCompatActivity() {
    private lateinit var userCourse: EditText

    //playedCourse
    override fun onCreate(savedInstanceState: Bundle?) {
        var userID = ""
        val sp = PreferenceManager.getDefaultSharedPreferences(applicationContext)
        val displayName = sp.getString( key: "displayName", defValue: "")
        val handicap = sp.getString( key: "handicap", defValue: "")
```

Figure 5.12: preference values being called and used in ScoreActivity

5.1.2 Gradle

The Gradle is a build system used for building an application. For Android, it is used to build, test and deploy the application to an Android device.

With Android, a Gradle folder is essential, due to it being used to generate an apk from the kt and XML files that have been written in the project. An apk is what renders an application to function on an Android device. The Gradle folder reads through all the files in the project and converts them into dex files before putting all the data into a single apk file.

The Gradle folder is located in the root directory. This folder contains all the dex and apk files that run the application. The project contains 2 build.gradle files, one located in the root directory, and the other is located inside the app folder. The root file applies what version of Kotlin the application is written in and all the dependencies of the application such as MongoDB Realm.

The file set in the app folder applies the android version associated with this application and all the dependencies that are needed to run that version of Android in this app. The ability to sync the application with MongoDB Realm is set to true here, this allows the app to read, write, update and delete collections stored in the MongoDB database.

5.2 Backend

With our application, we set up a MongoDB server to allow a user the ability to create an account to store their golf course results. To allow the user to access the MongoDB server we needed a register user page and a login page.

5.2.1 MongoDB

Realm

To connect to MongoDB, the application uses MongoDB Realm. This is an application on the server-side which allows the android application to access the MongoDB Atlas Cluster. The ability to access this only occurs for users who have registered. The email and password act as a way to verify an user to allow them to store their results inside the database, while the userID will act as a way to verify who is trying to connect to the data while also filter what data can be seen on the User History page.

Atlas

The MongoDB Atlas service is where the MongoDB database is located. A MongoDB Atlas cluster consists of a collection and a database. To connect to this database.

5.2.2 Integration

How the user can register for MongoDB Cluster Access

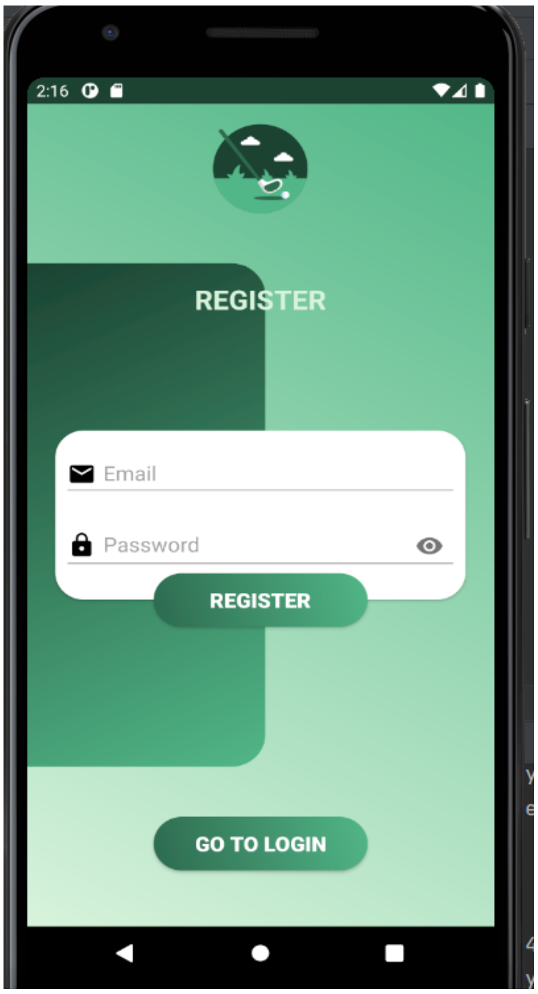
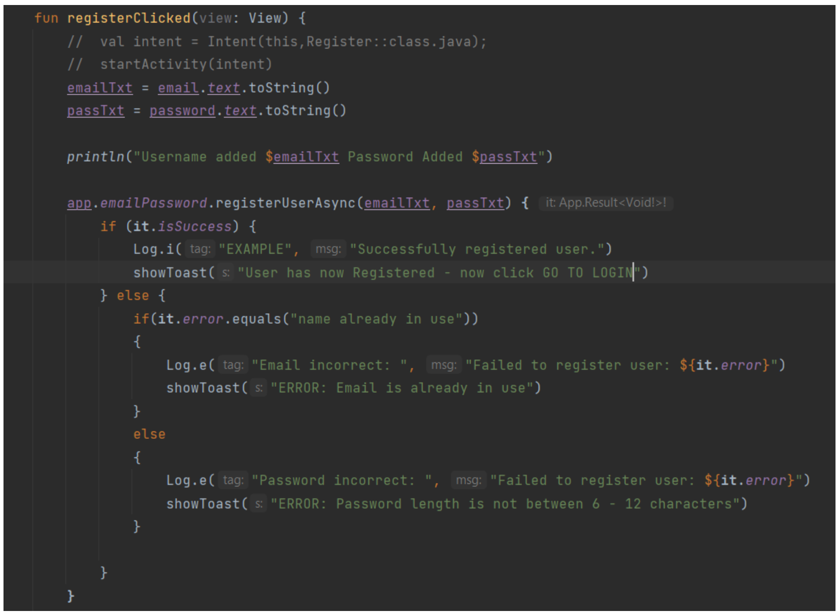


Figure 5.13: Page for registering a new user

The above image is a screenshot of the register page that is used to register a new user for MongoDB server access. To create a user the person who has opened the application needs to input an email and a password. If the user input has not been accepted the user gets an alert. That alert info will be different depending on what input error has occurred. If the user input is accepted an alert comes up saying: User has now registered - now click GO TO LOGIN.



```
fun registerClicked(view: View) {
    //  val intent = Intent(this,Register::class.java);
    //  startActivity(intent)
    emailTxt = email.text.toString()
    passTxt = password.text.toString()

    println("Username added $emailTxt Password Added $passTxt")

    app.emailPassword.registerUserAsync(emailTxt, passTxt) {  it: App.Result<Void!>!
        if (it.isSuccess) {
            Log.i( tag: "EXAMPLE",  msg: "Successfully registered user.")
            showToast( s: "User has now Registered - now click GO TO LOGIN")
        } else {
            if(it.error.equals("name already in use"))
            {
                Log.e( tag: "Email incorrect: ",  msg: "Failed to register user: ${it.error}")
                showToast( s: "ERROR: Email is already in use")
            }
            else
            {
                Log.e( tag: "Password incorrect: ",  msg: "Failed to register user: ${it.error}")
                showToast( s: "ERROR: Password length is not between 6 - 12 characters")
            }

        }
    }
```

Figure 5.14: Code associated with the register page

Above is an image that depicts the code that is used to register a new user for MongoDB server access. To create a user the person who has opened the application needs to input an email and password. If the user input has not been accepted the user gets an alert. That alert info will be different depending on what input error has occurred. If the user input is accepted an alert comes up saying: User has now registered - now click GO TO LOGIN. The GO TO LOGIN button brings the user to the login page to sign into the MongoDB server to access the user data that is attached to that user.
The user input fields are saved as emailTxt and passTxt string variables by utilizing the toString function in Kotlin. The string variables are passed into the function app.emailPassword.registerUserAsync() to be sent to the MongoDb Realm to create a new App user.

How the user can login to the MongoDB Cluster

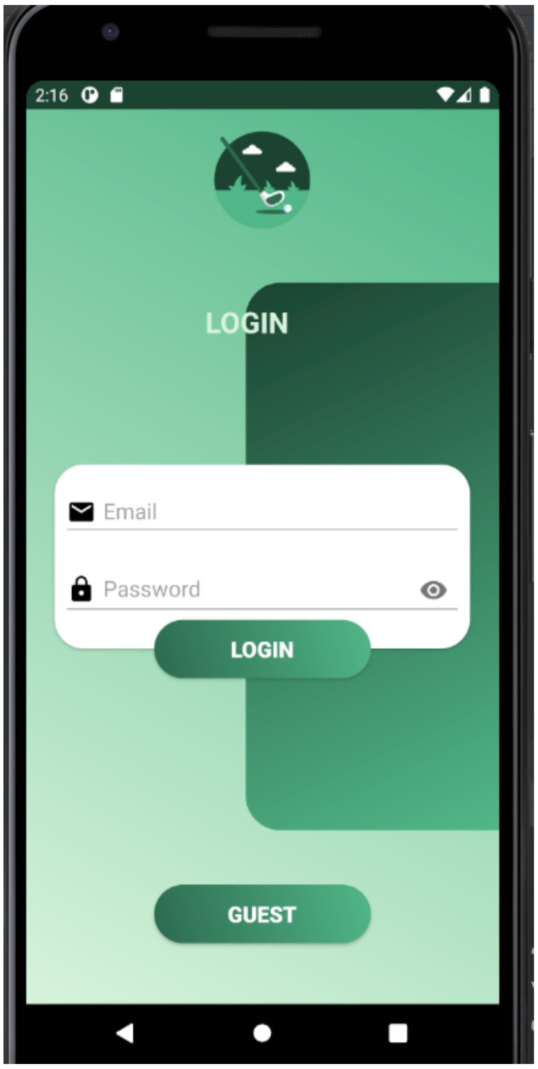
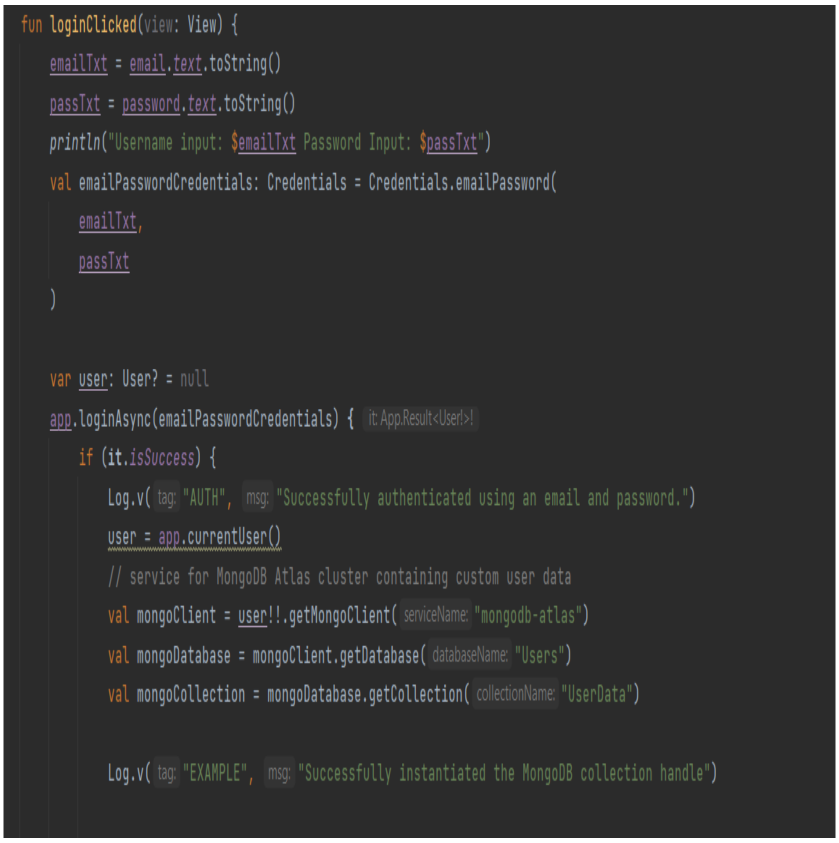


Figure 5.15: Page for logging in a user

To connect to the MongoDB server, the user needs to input their login details into the application when they are on the Login Page. The page can be seen in Figure 5.3 to the MongoDb Realm Application. When the user is on the login page it will enter its userID, email, and password again. The user will get an alert saying successfully logged in UserID.

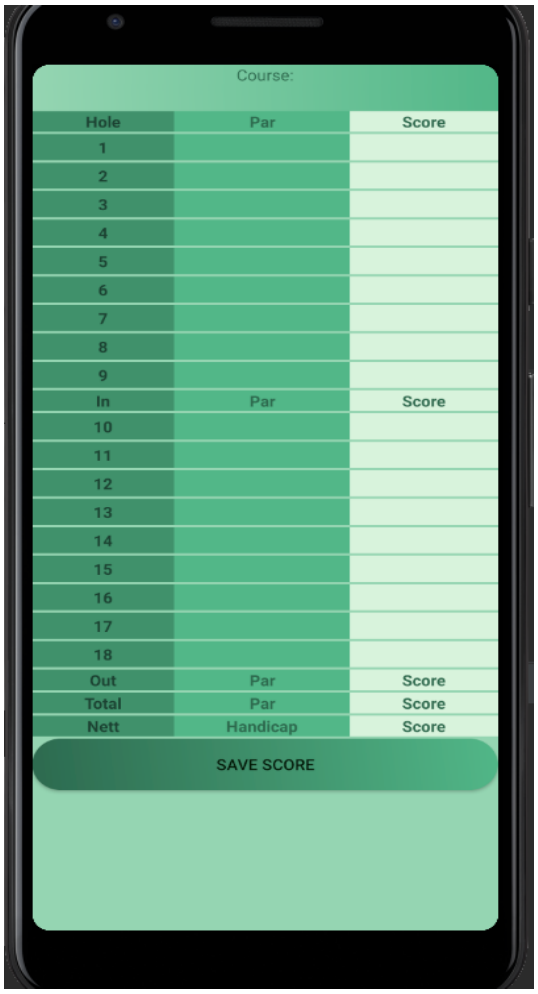


```
fun loginClicked(view: View) {
    emailTxt = email.text.toString()
    passTxt = password.text.toString()
    println("Username input: $emailTxt Password Input: $passTxt")
    val emailPasswordCredentials: Credentials = Credentials.emailPassword(
        emailTxt,
        passTxt
    )

    var user: User? = null
    app.loginAsync(emailPasswordCredentials) { it: App.Result<User!>!
        if (it.isSuccess) {
            Log.v( tag: "AUTH", msg: "Successfully authenticated using an email and password.")
            user = app.currentUser()
            // service for MongoDB Atlas cluster containing custom user data
            val mongoClient = user!!.getMongoClient( serviceName: "mongodb-atlas")
            val mongoDatabase = mongoClient.getDatabase( databaseName: "Users")
            val mongoCollection = mongoDatabase.getCollection( collectionName: "UserData")

            Log.v( tag: "EXAMPLE", msg: "Successfully instantiated the MongoDB collection handle")
```

Figure 5.16: Code associated with the login Page

By using the MongoDB loginAsync method for signing into the server we can get back an error if the sign-in was not successful and this method also allows the user to read and write to and from the database.
The application also gets back the data the user has stored up in the server. It finds the data by searching the database for a unique identifier. That unique identifier is the userID. The userID is the partition key for the MongoDB Atlas database, This allows the MongoDB database to be filtered for use in the user history page. The output is only the data that the current user of the application has sent to the MongoDB server.
If the user has not saved any data an alert will appear on the device with No User history Found outputted. The use of alerts may be minor but it has a big effect on the user experience. The ability to know what error you have when trying to run a specific function is key to solving that issue, this is where the use of alerts in the application is very effective.

With our application, we also send data to the server. That data is generated on the score page. The score page lets the user store results from a full round of golf including the course par score and user round score. The Nett score for the round of golf is generated by taking the handicap away from the user's total round score.

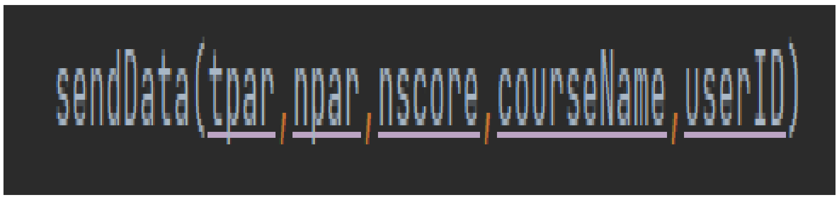

Course:

Hole	Par	Score
1		
2		
3		
4		
5		
6		
7		
8		
9		
In	Par	Score
10		
11		
12		
13		
14		
15		
16		
17		
18		
Out	Par	Score
Total	Par	Score
Nett	Handicap	Score

Figure 5.17: A snippet of the score page

When the user presses the save data button, the round is saved in the application before that saved data is sent to the MongoDB database to be stored in a collection.

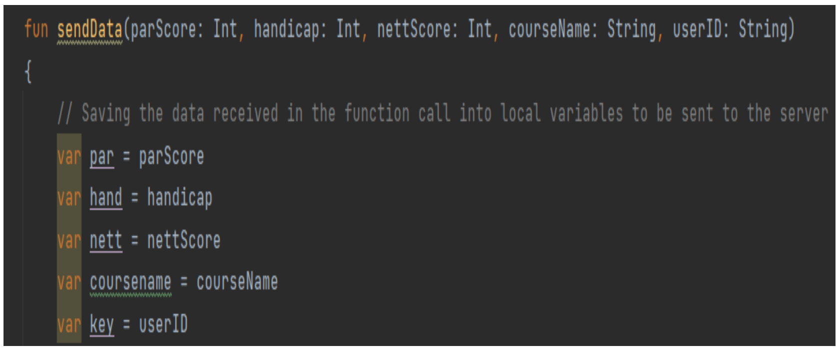
The image below illustrates how we are sending the data collected on this page to the server. It is done by calling the send data function.



```
sendData(tpar,npar,nscore,courseName,userID)
```

Figure 5.18: Calling the function for sending the data to the server

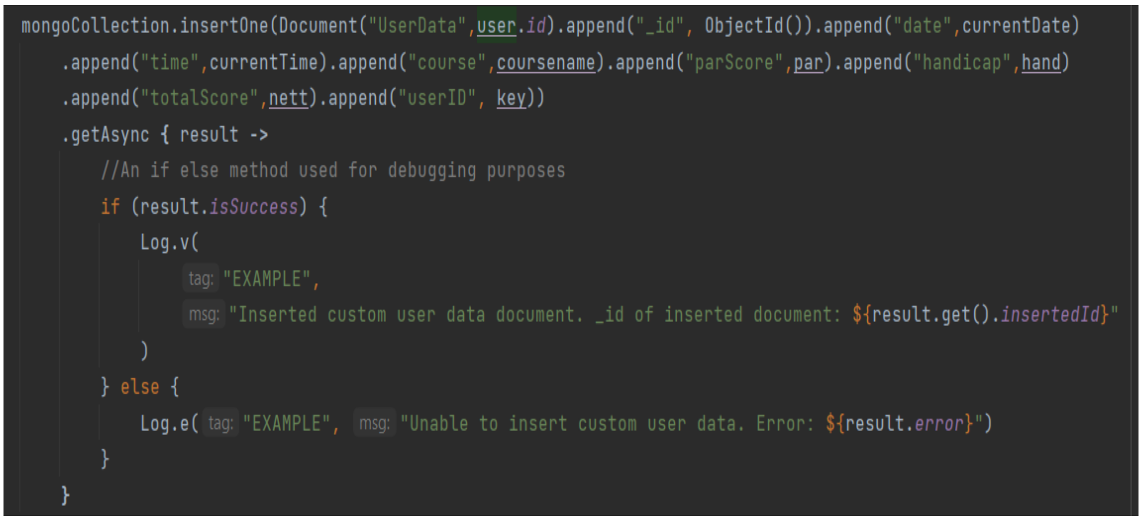
The values sent to the function are all processed in the ScoreActivity page. tpar is the course par score, npar is the course handicap that the user has assigned in the SettingsFragment page for that round, nscore is the users total score for the round when taking into consideration the handicap. The courseName is the name of the course the user has specified in the top textView in the fragment_score.xml and the userID is the unique username that the user has created for itself inside the SettingsFragment page.



```
fun sendData(parScore: Int, handicap: Int, nettScore: Int, courseName: String, userID: String)
{
    // Saving the data received in the function call into local variables to be sent to the server
    var par = parScore
    var hand = handicap
    var nett = nettScore
    var coursename = courseName
    var key = userID
```

Figure 5.19: The function used sending the data to the server

The image above shows how the function stores the data it receives from the function call to be used in the code shown below. We receive the data collected on the score page and save it in local variables that are used to send that data to the MongoDB server in the method illustrated below.



```
mongoCollection.insertOne(Document("UserData",user.id).append("_id", ObjectId()).append("date",currentDate)
    .append("time",currentTime).append("course",coursename).append("parScore",par).append("handicap",hand)
    .append("totalScore",nett).append("userID", key))
    .getAsync { result ->
        //An if else method used for debugging purposes
        if (result.isSuccess) {
            Log.v(
                tag: "EXAMPLE",
                msg: "Inserted custom user data document. _id of inserted document: ${result.get().insertedId}"
            )
        } else {
            Log.e( tag: "EXAMPLE",  msg: "Unable to insert custom user data. Error: ${result.error}")
        }
    }
```

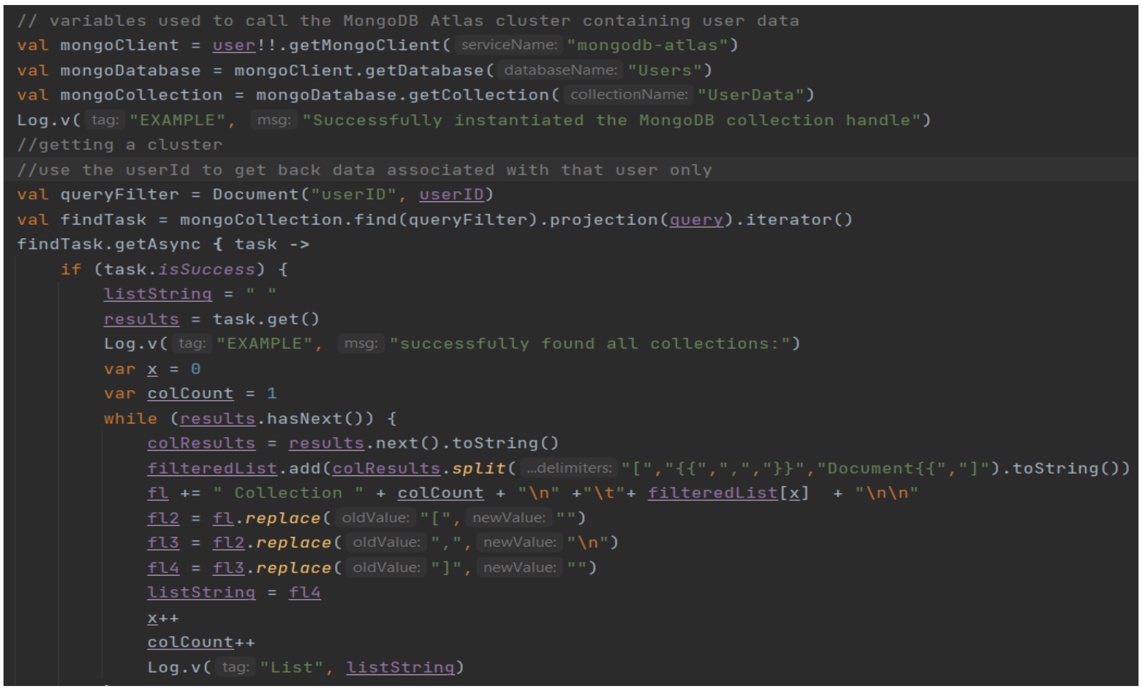
Figure 5.20: How the data is sent to the server

With MongoDB Realm, there are several different methods built into the system that can be used to perform CRUD operations. Crud operations stand for Create Read Update and Delete. These are the staples of any MySql or NoSql service. Without them, they would be useless. The function insertOne allows a user who has credentials to send the data to the server.

The collection has a name that needs to be called to start the process, our collection is called UserData. The use of append allows the creation of new values to be stored in fields already associated with the collection. The fields we have set up are id, date, time, course, parScore, handicap, totalScore, and userID. The data we send up to the server are the values received in the function call. Those values along with the current date and time.

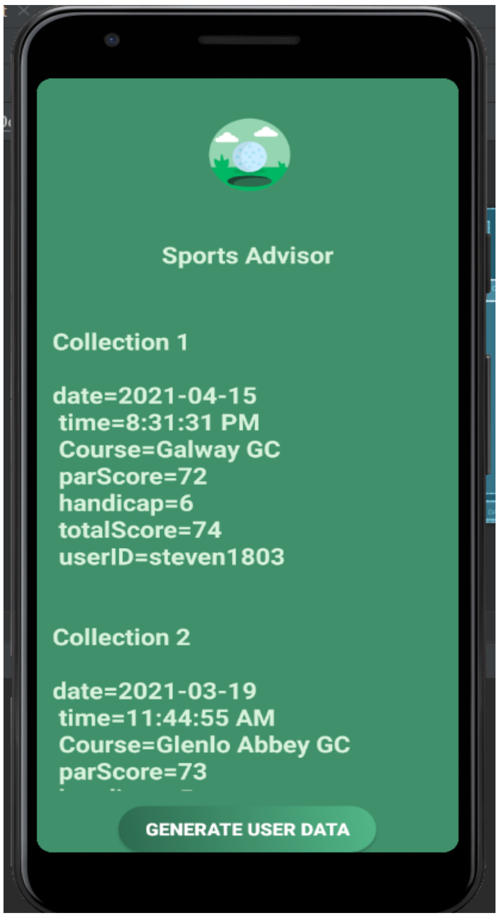
The date format is DD-MM-YYYY while the time format is Hours-Minutes-Seconds. Those 2 values are generated when the function is called for the exact date and time. They utilize the java SimpleDateFormat Class to generate the time and date in the format that we wanted and while giving back the correct results.

How the data received is outputted to the User



```
// variables used to call the MongoDB Atlas cluster containing user data
val mongoClient = user!!.getMongoClient( serviceName: "mongodb-atlas")
val mongoDatabase = mongoClient.getDatabase( databaseName: "Users")
val mongoCollection = mongoDatabase.getCollection( collectionName: "UserData")
Log.v( tag: "EXAMPLE",  msg: "Successfully instantiated the MongoDB collection handle")
//getting a cluster
//use the userId to get back data associated with that user only
val queryFilter = Document("userID", userID)
val findTask = mongoCollection.find(queryFilter).projection(query).iterator()
findTask.getAsync { task ->
    if (task.isSuccess) {
        listString = " "
        results = task.get()
        Log.v( tag: "EXAMPLE",  msg: "successfully found all collections:")
        var x = 0
        var colCount = 1
        while (results.hasNext()) {
            colResults = results.next().toString()
            filteredList.add(colResults.split( ...delimiters: "[","{{",",","}}","Document{{","]").toString())
            fl += " Collection " + colCount + "\n" +"\t"+ filteredList[x]  + "\n\n"
            fl2 = fl.replace( oldValue: "[", newValue: "")
            fl3 = fl2.replace( oldValue: ",", newValue: "\n")
            fl4 = fl3.replace( oldValue: "]", newValue: "")
            listString = fl4
            x++
            colCount++
            Log.v( tag: "List", listString)
```

Figure 5.21: Code snippet of how the data is sent to the server



```
Collection 1

date=2021-04-15
 time=8:31:31 PM
 Course=Galway GC
 parScore=72
 handicap=6
 totalScore=74
 userID=steven1803


Collection 2

date=2021-03-19
 time=11:44:55 AM
 Course=Glenlo Abbey GC
 parScore=73
```

Figure 5.22: How the data is sent to the server

The data is received and shown on the User History Page. When the page is opened the MongoDB collection is called and the collection data is saved in a string. This string is then outputted to the app when the Generate User Data Button is pressed.
The page has an onCreate function that contains the code in Figure 5.9. The code allows the MongoDB database to be filtered by the userID to output only the data that is associated with that id key. The userID is set on the settings page. This is how we protect the user's data and stop unauthorized access to data stored on the database. All userId values are unique. The data is outputted in order of oldest stored data to the latest stored data. The outputted data is the date, time, Course, parScore, handicap, totalScore, and userID with a line separated between each output for better readability.

5.3 Data Gathering and Processing

For our application, one of the major parts of the project was to retrieve weather data from a weather API using the OKHTTP: HTTP and HTTP2 client[1]. This data is then gathered and then parsed from Strings and Objects to their JSON representations using GSON[23]. This data is then processed and after reading in all the data, a rating between 1 and 5 is then applied to an hour associated with the weather values passed in.

AccuWeather

One of the main aspects is retrieving data from the AccuWeather API[14], this was fundamental part of the app as without it, hourly ratings based on conditions for playing golf couldnt be shown to the user for their chosen golf course. We decided upon implementing Accuweather as part of our Application as it was able to search for locations based on their own specific location codes, which then allowed us to create specific locations of golf courses in Co. Galway on a list for the user to pick from.

OKHTTP

For our data retrieval we used Okhttp to gather the weather data from the Accuweather API,we decided upon this as the technology to retrieving data because of one its main features, its perseverance in poor network conditions or when the network is being troublesome: ”it will silently recover from common connection problems. If your service has multiple IP addresses, OkHttp will attempt alternate addresses if the first connect fails.”[1]

This is vitally important to the design of our Application as we know that not every golf course will have good WiFi at their clubs and then when the player is on the golf course they may have to use their mobile data and in certain locations, the mobile data may be poor or non-existent so by using this technology, it will still try to make these requests even under the poorest of conditions.

5.3.1 GSON

After OKHTTP retrieves the data back from the Accuweather API service it returns back string values and objects that need to be parsed into JSON Representations of that data so that it can be processed into ratings, this is where GSON comes into play, it takes the data and converts it to JSON readable data, this is then stored into multiple Kotlin data class Files which can then be called upon to calculate the ratings for the hour.

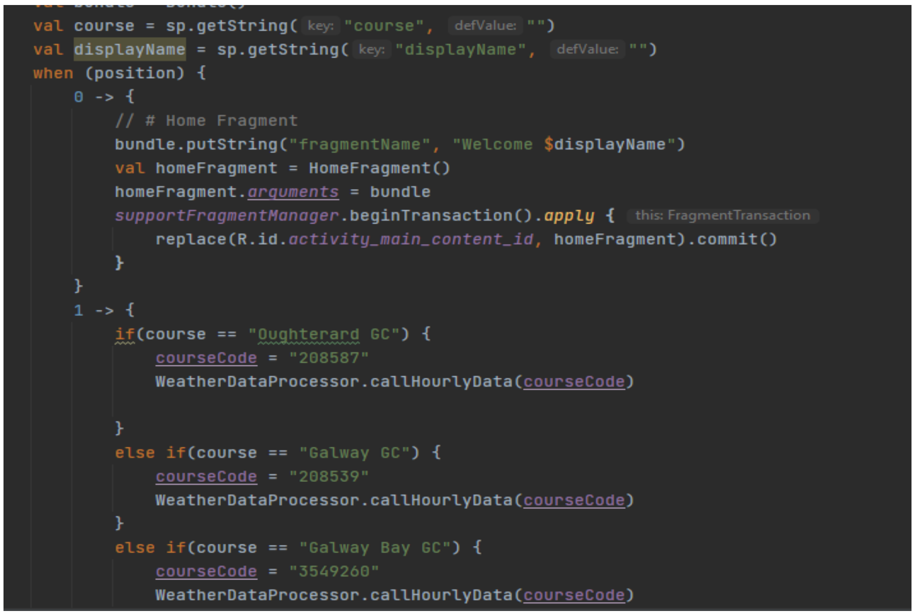
5.3.2 Processing the Data

When the data has converted from a String to its JSON Representation, it is then time to take that data and compare it to predetermined values which will check to see if data that is being passed through fits inside any of of the ranges of values it will be given a value between 0 and 1.

There are six different types of data that are passed through to the data processor, these include Rainfall, windSpeed, temperature, real Feel temperature, Humidity, and if the hour is in daylight. The first five values determine the rating out of 5 and each one of these values are rated between 0 and 1 to give a more accurate rating. The final value, the isDayLight value, then checks to see if the hour is in daylight or not, if it is in daylight, the rating remains the same otherwise the rating for that hour is set to 0.0 as it is not recommended to play golf in the dark.

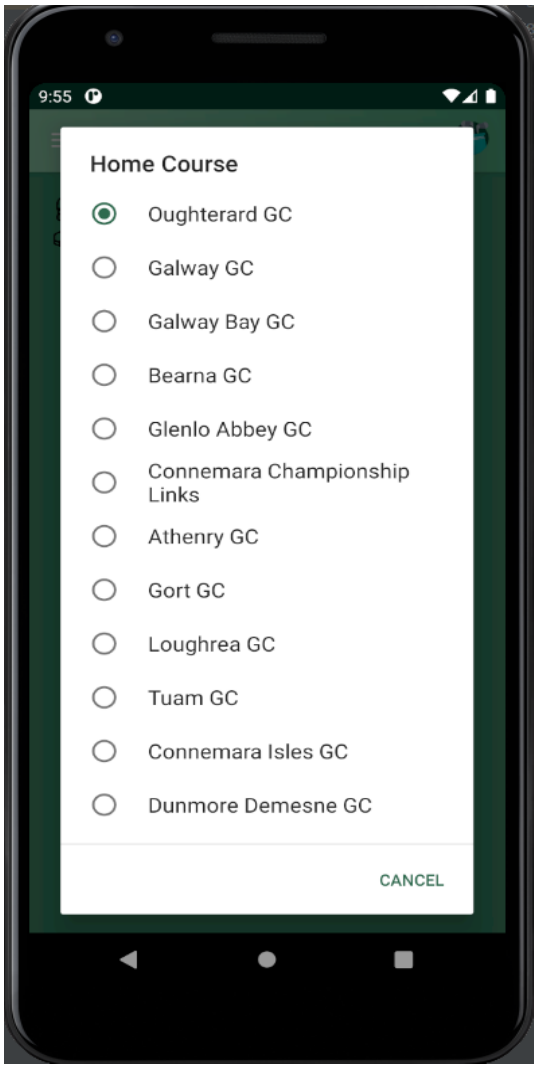
5.3.3 Integration

All of these sections integrate and work together to produce a rating for the hour for the user to then determine which time would be best to start their round of golf, below will show how the data is retrieved, processed, and outputted to the user.



```
val course = sp.getString( key: "course",  defValue: "")
val displayName = sp.getString( key: "displayName",  defValue: "")
when (position) {
    0 -> {
        // # Home Fragment
        bundle.putString("fragmentName", "Welcome $displayName")
        val homeFragment = HomeFragment()
        homeFragment.arguments = bundle
        supportFragmentManager.beginTransaction().apply {  this: FragmentTransaction
            replace(R.id.activity_main_content_id, homeFragment).commit()
        }
    }
    1 -> {
        if(course == "Oughterard GC") {
            courseCode = "208587"
            WeatherDataProcessor.callHourlyData(courseCode)

        }
        else if(course == "Galway GC") {
            courseCode = "208539"
            WeatherDataProcessor.callHourlyData(courseCode)
        }
        else if(course == "Galway Bay GC") {
            courseCode = "3549260"
            WeatherDataProcessor.callHourlyData(courseCode)
```

Figure 5.23: Course Codes used for API Request

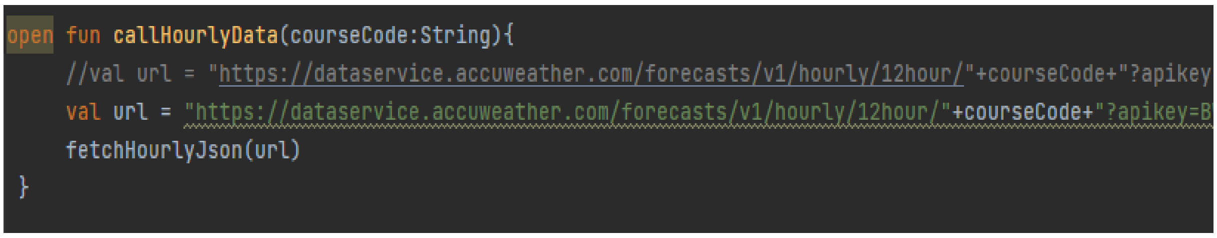


Home Course

- [✓ Oughterard GC]
- Galway GC
- Galway Bay GC
- Bearna GC
- Glenlo Abbey GC
- Connemara Championship Links
- Athenry GC
- Gort GC
- Loughrea GC
- Tuam GC
- Connemara Isles GC
- Dunmore Demesne GC

Figure 5.24: List of Courses used for Location

The above images show how based on the location the user has set as their chosen golf course, it will set the course code to that golf course and will then begin the first part in trying to retrieve the weather data for that particular golf course regardless of where the user is located.
Once the course has been selected, the course code is then passed to our callHourlyData Method which changes the course code for the API call the invokes the fetchCurrentJson method which then fetches the data.

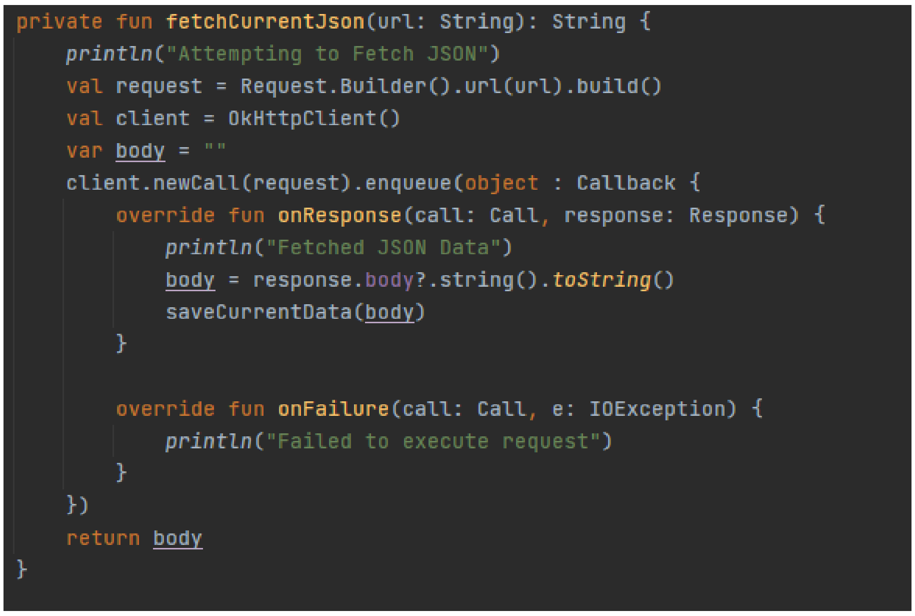


```kotlin
open fun callHourlyData(courseCode:String){
    //val url = "https://dataservice.accuweather.com/forecasts/v1/hourly/12hour/"+courseCode+"?apikey
    val url = "https://dataservice.accuweather.com/forecasts/v1/hourly/12hour/"+courseCode+"?apikey=B
    fetchHourlyJson(url)
}
```

Figure 5.25: Course Codes used for API Request

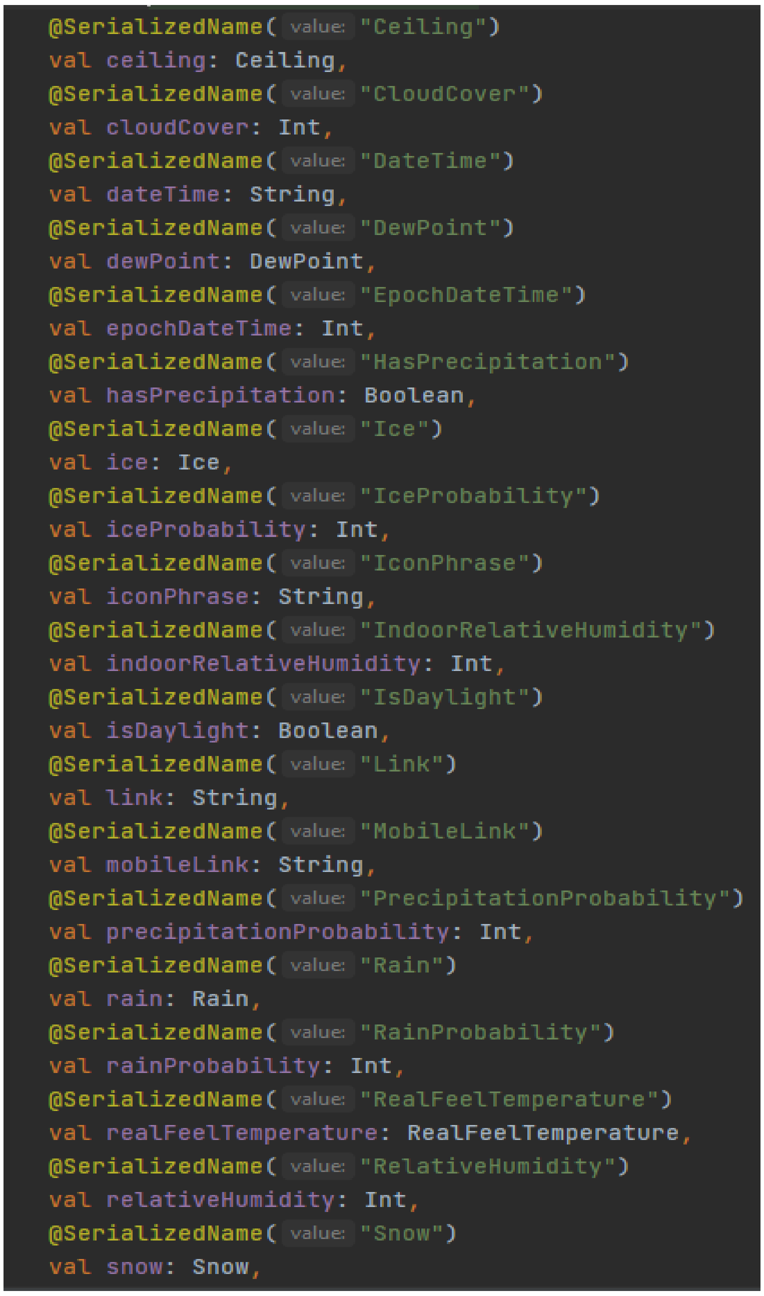
After the course code has been set, the data is then fetched using okhttp to fetch the JSON data from the API, while being fetched, the data is also stored locally using GSON in order to output the data to the user.

This data is stored in a Kotlin data class files in order for the specific data required for the hourly or current hour ratings to be created. By doing this it allows us to call each data type independently from the other data types so that we only use what we need to use.



```kotlin
private fun fetchCurrentJson(url: String): String {
    println("Attempting to Fetch JSON")
    val request = Request.Builder().url(url).build()
    val client = OkHttpClient()
    var body = ""
    client.newCall(request).enqueue(object : Callback {
        override fun onResponse(call: Call, response: Response) {
            println("Fetched JSON Data")
            body = response.body?.string().toString()
            saveCurrentData(body)
        }

        override fun onFailure(call: Call, e: IOException) {
            println("Failed to execute request")
        }
    })
    return body
}
```

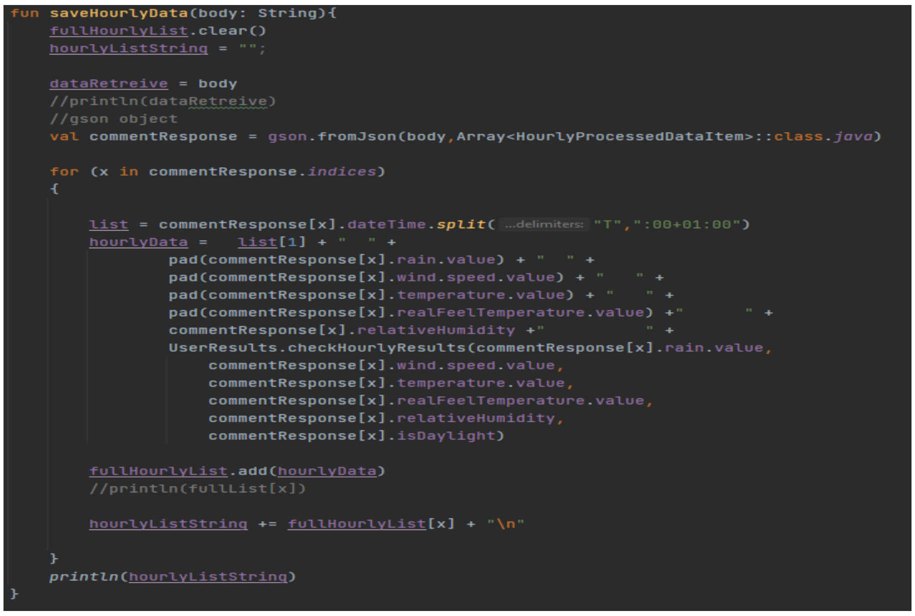
Figure 5.26: Fetching of JSON Data using OKHTTP



```
@SerializedName( value: "Ceiling")
val ceiling: Ceiling,
@SerializedName( value: "CloudCover")
val cloudCover: Int,
@SerializedName( value: "DateTime")
val dateTime: String,
@SerializedName( value: "DewPoint")
val dewPoint: DewPoint,
@SerializedName( value: "EpochDateTime")
val epochDateTime: Int,
@SerializedName( value: "HasPrecipitation")
val hasPrecipitation: Boolean,
@SerializedName( value: "Ice")
val ice: Ice,
@SerializedName( value: "IceProbability")
val iceProbability: Int,
@SerializedName( value: "IconPhrase")
val iconPhrase: String,
@SerializedName( value: "IndoorRelativeHumidity")
val indoorRelativeHumidity: Int,
@SerializedName( value: "IsDaylight")
val isDaylight: Boolean,
@SerializedName( value: "Link")
val link: String,
@SerializedName( value: "MobileLink")
val mobileLink: String,
@SerializedName( value: "PrecipitationProbability")
val precipitationProbability: Int,
@SerializedName( value: "Rain")
val rain: Rain,
@SerializedName( value: "RainProbability")
val rainProbability: Int,
@SerializedName( value: "RealFeelTemperature")
val realFeelTemperature: RealFeelTemperature,
@SerializedName( value: "RelativeHumidity")
val relativeHumidity: Int,
@SerializedName( value: "Snow")
val snow: Snow,
```

Figure 5.27: Example of the Data set used to store weather values

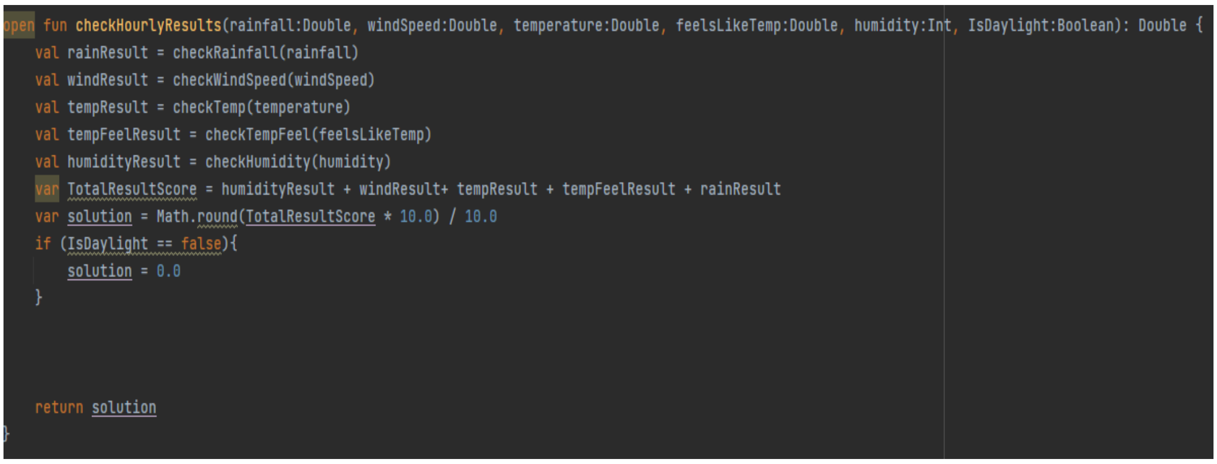
While saving the data locally this is where the function for calculating the ratings is called and converted to a list along with all the data values being displayed to the user(Rainfall, WindSpeed, Temperature, Feels Like Temperature, Humidity, Hourly Rating).



```
fun saveHourlyData(body: String){
    fullHourlyList.clear()
    hourlyListString = "";

    dataRetreive = body
    //println(dataRetreive)
    //gson object
    val commentResponse = gson.fromJson(body,Array<HourlyProcessedDataItem>::class.java)

    for (x in commentResponse.indices)
    {

        list = commentResponse[x].dateTime.split( ...delimiters: "T",":00+01:00")
        hourlyData =   list[1] + "   " +
                pad(commentResponse[x].rain.value) + "   " +
                pad(commentResponse[x].wind.speed.value) + "    " +
                pad(commentResponse[x].temperature.value) + "    " +
                pad(commentResponse[x].realFeelTemperature.value) +"       " +
                commentResponse[x].relativeHumidity +"          " +
                UserResults.checkHourlyResults(commentResponse[x].rain.value,
                    commentResponse[x].wind.speed.value,
                    commentResponse[x].temperature.value,
                    commentResponse[x].realFeelTemperature.value,
                    commentResponse[x].relativeHumidity,
                    commentResponse[x].isDaylight)

        fullHourlyList.add(hourlyData)
        //println(fullList[x])

        hourlyListString += fullHourlyList[x] + "\n"

    }
    println(hourlyListString)
}
```

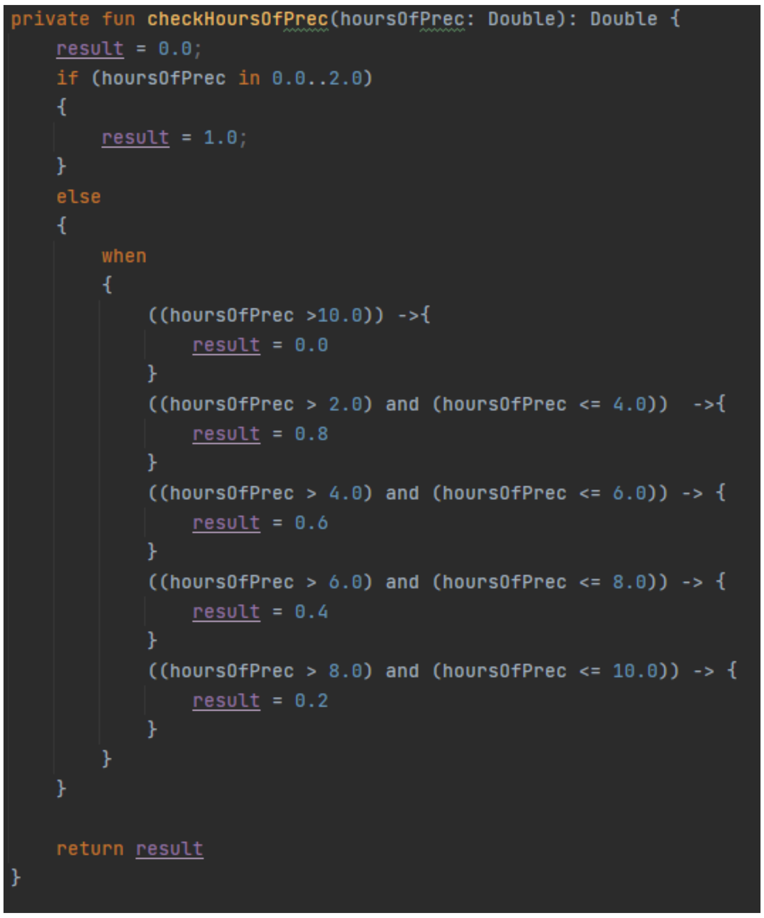
Figure 5.28: Storing of JSON data to Dataset Files

The function takes in all the above values except for Hourly Rating and a value for if it is daytime and then compares them to a set of predetermined values for each of the five values, each one is given a rating between 0 and 1 which then will give a rating between 0.0 and 5.0 unless the hour is passed sunset or before sunrise.



```
open fun checkHourlyResults(rainfall:Double, windSpeed:Double, temperature:Double, feelsLikeTemp:Double, humidity:Int, IsDaylight:Boolean): Double {
    val rainResult = checkRainfall(rainfall)
    val windResult = checkWindSpeed(windSpeed)
    val tempResult = checkTemp(temperature)
    val tempFeelResult = checkTempFeel(feelsLikeTemp)
    val humidityResult = checkHumidity(humidity)
    var TotalResultScore = humidityResult + windResult+ tempResult + tempFeelResult + rainResult
    var solution = Math.round(TotalResultScore * 10.0) / 10.0
    if (IsDaylight == false){
        solution = 0.0
    }



    return solution
}
```

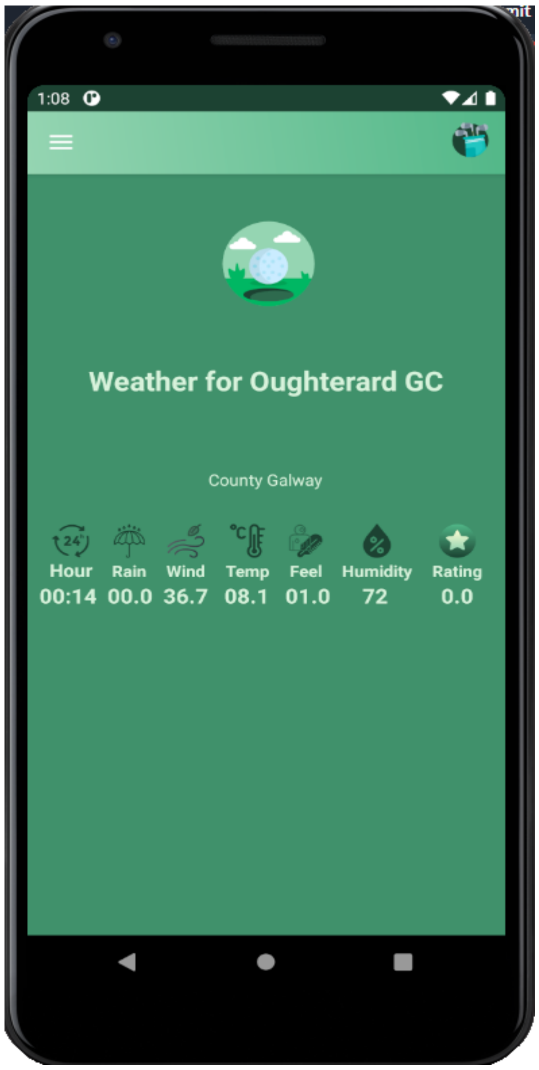
Figure 5.29: Calculation of the rating



```kotlin
private fun checkHoursOfPrec(hoursOfPrec: Double): Double {
    result = 0.0;
    if (hoursOfPrec in 0.0..2.0)
    {
        result = 1.0;
    }
    else
    {
        when
        {
            ((hoursOfPrec >10.0)) ->{
                result = 0.0
            }
            ((hoursOfPrec > 2.0) and (hoursOfPrec <= 4.0))  ->{
                result = 0.8
            }
            ((hoursOfPrec > 4.0) and (hoursOfPrec <= 6.0)) -> {
                result = 0.6
            }
            ((hoursOfPrec > 6.0) and (hoursOfPrec <= 8.0)) -> {
                result = 0.4
            }
            ((hoursOfPrec > 8.0) and (hoursOfPrec <= 10.0)) -> {
                result = 0.2
            }
        }
    }

    return result
}
```

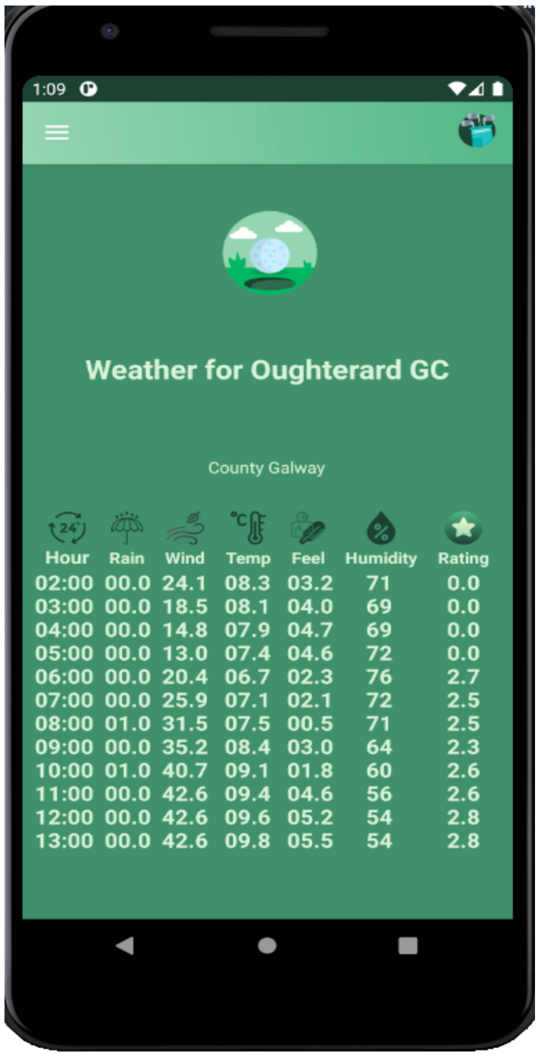
Figure 5.30: Example of how the data is compared

After all the data is processed it is then outputted to the user so that they can see the ratings for the next twelve hours or the current hour.



Hour	Rain	Wind	Temp	Feel	Humidity	Rating
00:14	00.0	36.7	08.1	01.0	72	0.0

Figure 5.31: Current Hour Weather Data



Weather for Oughterard GC

County Galway

Hour	Rain	Wind	Temp	Feel	Humidity	Rating
02:00	00.0	24.1	08.3	03.2	71	0.0
03:00	00.0	18.5	08.1	04.0	69	0.0
04:00	00.0	14.8	07.9	04.7	69	0.0
05:00	00.0	13.0	07.4	04.6	72	0.0
06:00	00.0	20.4	06.7	02.3	76	2.7
07:00	00.0	25.9	07.1	02.1	72	2.5
08:00	01.0	31.5	07.5	00.5	71	2.5
09:00	00.0	35.2	08.4	03.0	64	2.3
10:00	01.0	40.7	09.1	01.8	60	2.6
11:00	00.0	42.6	09.4	04.6	56	2.6
12:00	00.0	42.6	09.6	05.2	54	2.8
13:00	00.0	42.6	09.8	05.5	54	2.8

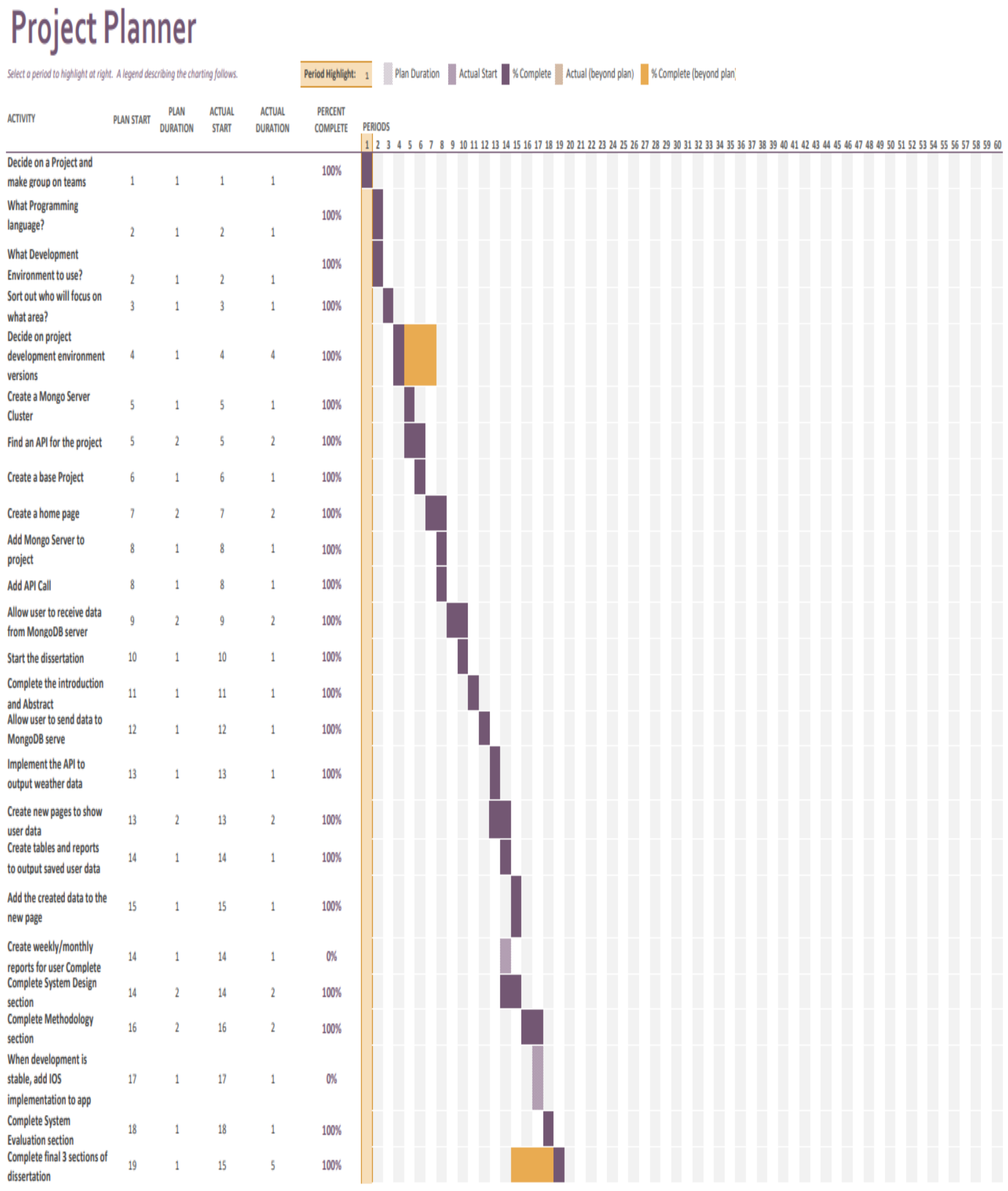
Figure 5.32: Next 12 Hours Weather Data

Chapter 6

System Evaluation

In this chapter, we focused on the Testing, Evaluation, and Limitations of our Project. Our focus was to test how robust or project is by following the Agile methodology of Software Testing. We designed a Gantt Chart to outline sprints that break down the project into smaller parts which in turn, would be vigorously tested. The full Gantt chart can be viewed on the link below in our Github Repository.

`https://github.com/stevenJoyce/4thYearGroupProject/tree/main/TestingGuides`



Project Planner

Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 1

Plan Duration | Actual Start | % Complete | Actual (beyond plan) | % Complete (beyond plan)

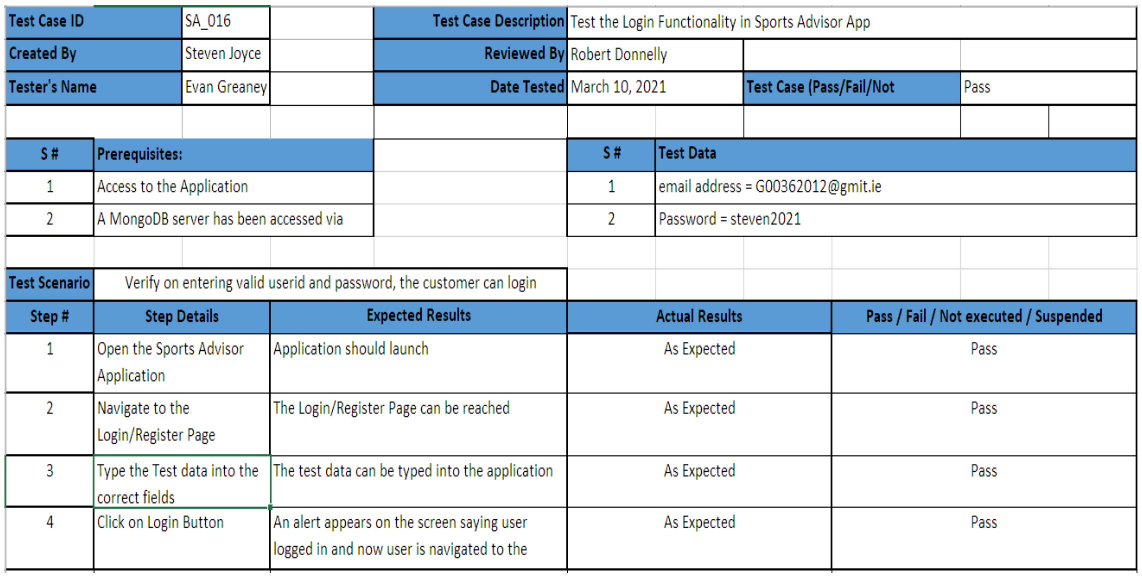
ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Decide on a Project and make group on teams	1	1	1	1	100%
What Programming language?	2	1	2	1	100%
What Development Environment to use?	2	1	2	1	100%
Sort out who will focus on what area?	3	1	3	1	100%
Decide on project development environment versions	4	1	4	4	100%
Create a Mongo Server Cluster	5	1	5	1	100%
Find an API for the project	5	2	5	2	100%
Create a base Project	6	1	6	1	100%
Create a home page	7	2	7	2	100%
Add Mongo Server to project	8	1	8	1	100%
Add API Call	8	1	8	1	100%
Allow user to receive data from MongoDB server	9	2	9	2	100%
Start the dissertation	10	1	10	1	100%
Complete the introduction and Abstract	11	1	11	1	100%
Allow user to send data to MongoDB serve	12	1	12	1	100%
Implement the API to output weather data	13	1	13	1	100%
Create new pages to show user data	13	2	13	2	100%
Create tables and reports to output saved user data	14	1	14	1	100%
Add the created data to the new page	15	1	15	1	100%
Create weekly/monthly reports for user Complete	14	1	14	1	0%
Complete System Design section	14	2	14	2	100%
Complete Methodology section	16	2	16	2	100%
When development is stable, add IOS implementation to app	17	1	17	1	0%
Complete System Evaluation section	18	1	18	1	100%
Complete final 3 sections of dissertation	19	1	15	5	100%

Figure 6.1: Test Case for successful Login

6.1 Unit Testing

For our project we have used the Agile methodology to layout our plan. With this, testing is a major aspect of this methodology and as such, we created Test cases for each sprint and assigned a role for each team member. The roles assigned were as followed, Steven Joyce would create the unit tests which would be reviewed by Robert Donnelly before being implemented by Evan Greaney. These roles would remain unchanged throughout the testing process.

The illustration below is an example of a test case used for running a successful login attempt. As seen below each case has an ID that is linked to the Gantt chart. The case uses the test data provided and requires prerequisites to be met for the test to be attempted. The test must follow the steps in a linear order for the case to be successful.



Test Case ID	SA_016	Test Case Description	Test the Login Functionality in Sports Advisor App		
Created By	Steven Joyce	Reviewed By	Robert Donnelly		
Tester's Name	Evan Greaney	Date Tested	March 10, 2021	Test Case (Pass/Fail/Not	Pass

S #	Prerequisites:
1	Access to the Application
2	A MongoDB server has been accessed via

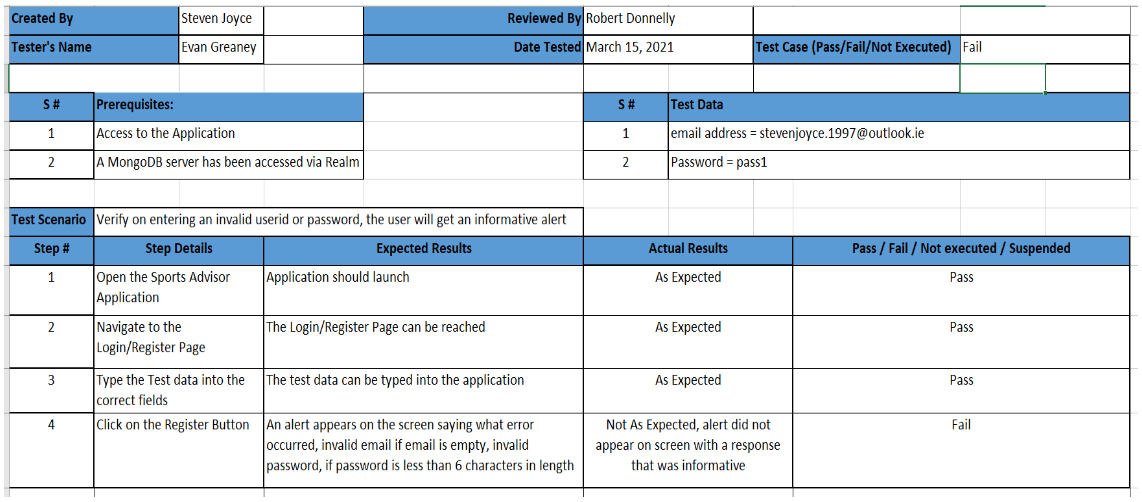
S #	Test Data
1	email address = G00362012@gmit.ie
2	Password = steven2021

Test Scenario	Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open the Sports Advisor Application	Application should launch	As Expected	Pass
2	Navigate to the Login/Register Page	The Login/Register Page can be reached	As Expected	Pass
3	Type the Test data into the correct fields	The test data can be typed into the application	As Expected	Pass
4	Click on Login Button	An alert appears on the screen saying user logged in and now user is navigated to the	As Expected	Pass

Figure 6.2: Test Case for successful Login

Some test cases are designed to confirm an unsuccessful outcome from a user's perspective. This can be seen in the image below, this test case although it passed most steps it failed to alert the user as to why they unsuccessfully logged in as the alert did not appear and ultimately the case failed. This case allowed us to implement alerts to summon at the correct time and place.



Created By	Steven Joyce	**Reviewed By**	Robert Donnelly
Tester's Name	Evan Greaney	**Date Tested**	March 15, 2021
Test Case (Pass/Fail/Not Executed)	Fail		

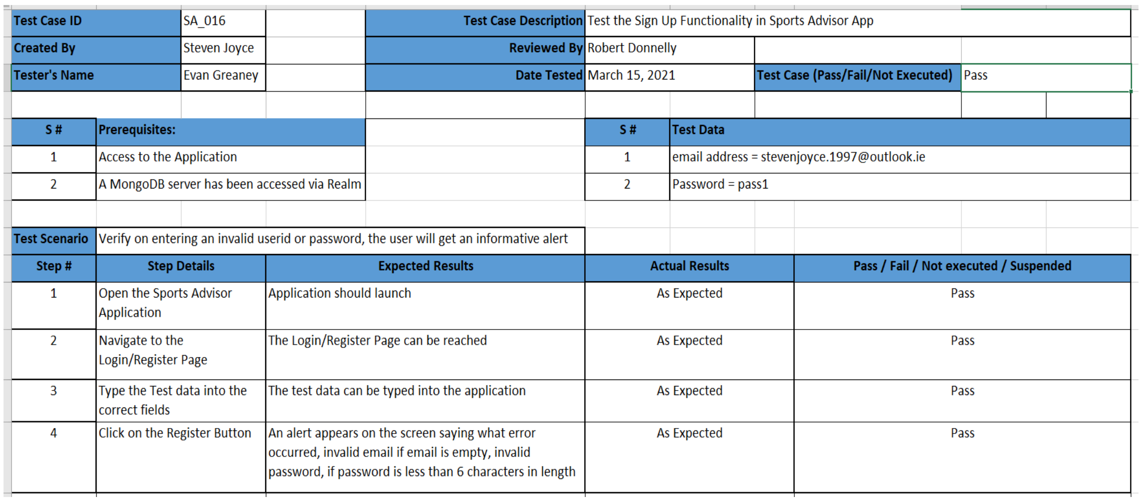
S #	**Prerequisites:**	**S #**	**Test Data**
1	Access to the Application	1	email address = stevenjoyce.1997@outlook.ie
2	A MongoDB server has been accessed via Realm	2	Password = pass1

Test Scenario Verify on entering an invalid userid or password, the user will get an informative alert

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Open the Sports Advisor Application	Application should launch	As Expected	Pass
2	Navigate to the Login/Register Page	The Login/Register Page can be reached	As Expected	Pass
3	Type the Test data into the correct fields	The test data can be typed into the application	As Expected	Pass
4	Click on the Register Button	An alert appears on the screen saying what error occurred, invalid email if email is empty, invalid password, if password is less than 6 characters in length	Not As Expected, alert did not appear on screen with a response that was informative	Fail

Figure 6.3: Test Case for unsuccessful Login

With an understanding of what went wrong in the above test case, we implemented changes to our code that would give the user an informative alert that helps understand what error occurred. The screenshot below is a test case for the modified code with a successful outcome.



Test Case ID	SA_016	**Test Case Description**	Test the Sign Up Functionality in Sports Advisor App
Created By	Steven Joyce	**Reviewed By**	Robert Donnelly
Tester's Name	Evan Greaney	**Date Tested**	March 15, 2021
Test Case (Pass/Fail/Not Executed)	Pass		

S #	**Prerequisites:**	**S #**	**Test Data**
1	Access to the Application	1	email address = stevenjoyce.1997@outlook.ie
2	A MongoDB server has been accessed via Realm	2	Password = pass1

Test Scenario Verify on entering an invalid userid or password, the user will get an informative alert

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Open the Sports Advisor Application	Application should launch	As Expected	Pass
2	Navigate to the Login/Register Page	The Login/Register Page can be reached	As Expected	Pass
3	Type the Test data into the correct fields	The test data can be typed into the application	As Expected	Pass
4	Click on the Register Button	An alert appears on the screen saying what error occurred, invalid email if email is empty, invalid password, if password is less than 6 characters in length	As Expected	Pass

Figure 6.4: Test Case for unsuccessful Login with Alerts

We did this type of testing for every feature of our application. This allowed us to debug our application regularly and efficiently. To see all of our test cases go to: `https://github.com/stevenJoyce/4thYearGroupProject/tree/main/TestingGuides`

6.2 Objectives Overview

Find a new programming language to learn independently

The group's goal of learning a new programming language was achieved as the team gained skills and knowledge of Kotlin JVM, which the team members have no prior experience with this development tool before the past college year.

Find a Methodology to best implement our idea

Upon starting the year the team had chosen two main methodologies to consider, being Agile and Waterfall. Due to Agile being a better fit for our application, the team decided to pick this methodology as our preferred approach.

Implement the methodology to create a project plan

With our methodology choice, the way we approached our project plan was to split up the work into manageable parts, these were our sprints. The sprints were then handed out to each member of the team to complete within a given deadline of a set amount of weeks. The team had successfully created a plan. At the end of the planning process, a Gantt chart was created to be used to assign sprints to each member and manage the progress of the application's development.

Find an architecture structure suitable to use

With Kotlin being our language of choice, upon researching, the team had discovered that it was mainly utilised for Android Development and that JetBrains, the company that is developing Kotlin also develops the IntelliJ IDE which supports the Android Studio SDK for android development. With this in mind, the team quickly and successfully came to an agreement that the application should be made in the JetBrains toolkit environment of IntelliJ and Android Studio.

Utilize an environment for stable version control

With our history of using GitHub it was a simple choice for the team to pick it as most developers struggle with learning a new version control environment so the team decided to stick with what they already had prior experience with.

With the project on GitHub we were able to set up an interactive Issue board. This allowed issues to be assigned to individual team members and for the issue's progress to be tracked. It allowed each group member to access the latest version of the application from any machine which allowed for remote working, this was a vital aid for our team in a covid affected work environment.

Establish a well structured base for the project

To design and implement our Android application we needed a great IDE. Through our GitHub Student Developer Pack, we were given a free license for IntelliJ Ultimate. By using this IDE we could create an emulator to test our application and call our MongoDB database to view any requests the application makes in real-time.

Create a high quality front-end

With Android Studio ADK being built into IntelliJ, we could design the page and write the code at the same time which helped us understand what we were trying to achieve for the end-user. We used a consistent colour scheme throughout the application XML layouts and drawable assets which resulted in a professional and appealing look and feel.

We altered XML icons which would have not suited the application's look but through applying the colour scheme we made them match our application's theme. Every aspect from the drawer layout to buttons in one way or another conforms to the colour scheme of the application which in turn resulted in a high-quality front-end design.

Find and deploy an API which our app can utilize

After multiple attempts at using different APIs for our application, we then decided on using the Accuweather API service. This allowed us to generate current weather conditions and the next 12 hours for the chosen area in Galway.

Create a server service for the Application.

With our prior knowledge and experience with MongoDB, it was a simple choice to make. We decided on using 2 aspects of MongoDB which are Atlas and Realm.

Allow for user data to be protected and appropriately stored

MongoDB Realm created a buffer between the server and application to filter unauthorized access to the user data. This a hidden layer of protection that filtered out any issues or unwarranted data.

Use of a Location service to find user's desired location

This objective was modified due to the API service, where the location can only be found using their custom location codes that a normal user would not have prior knowledge of. We changed into a list of all golf courses in Galway and allowed the user to save a preferred golf course.

Implement a login service which allows user to access their protected data

To view any user data, they have to sign to the application on the login page before setting their userID in the settings page. When the user goes to load its history they only view their own data.

Design the Apps software to successfully implement the apps intended functionality

The features that we originally set out to have on our application have all been achieved but the end result did not generate the original design plan we had intended due to limitations and a better understanding of what we wanted to achieve.

Deploy our methodologies style of testing to ensure quality standards are met

With the methodology testing approach we used, it favoured constant testing throughout the development cycle and as such allowed the team to fix unforeseen bugs and errors in our design and implementation as they happen. As developers, we tested our work after every change to the application with the built-in emulator. This ensured that we never committed non-functioning code to our repository.

6.3 Issues, limitations

This section mainly pertains to the Issues and Limitations experienced by the team throughout the whole development lifecycle of the Application. Due to the nature of our project development tools and language being relatively new to ourselves and the wider coding community, there have been many challenges we have faced to develop this application that has needed to be resolved.

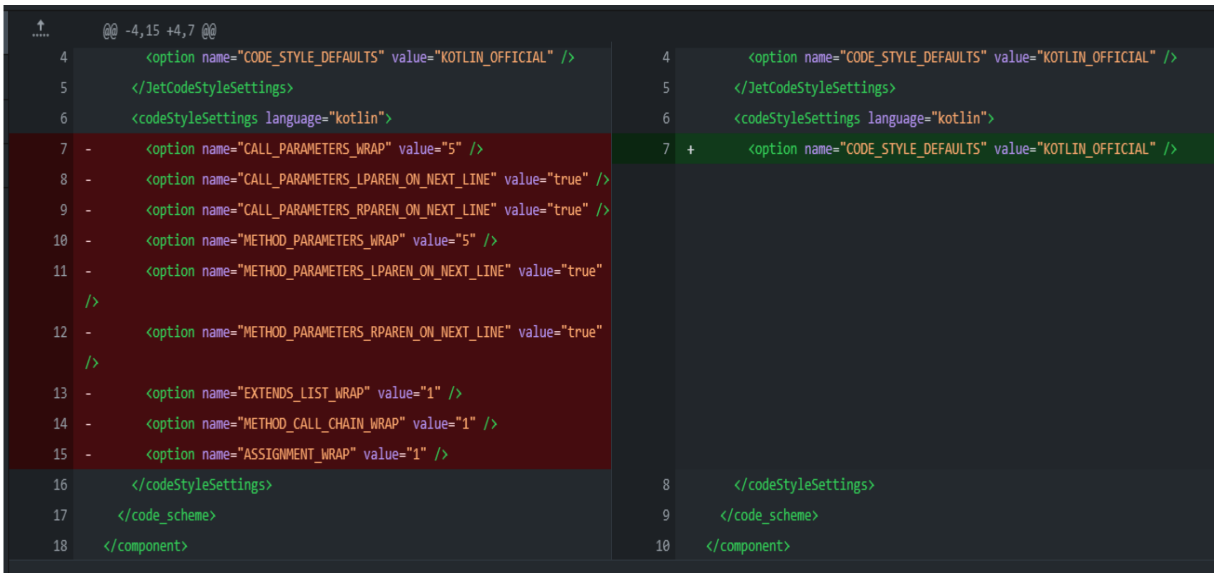
6.3.1 IntelliJ

Limitations

Our IDE only caused one limitation in the last stages of our development lifecycle when the IDE updated the Kotlin Language version, it forced updates upon our own project without consent and caused issues throughout all our .kt classes which led them to not be able to be rendered by the Gradle.

This resulted in us losing about 3 hours of development time due to us not realising what the root cause of the issue was. To fix this we had to look at our commit history through GitHub desktop to see what was changed file by file until we found the issue.

To resolve this we had to revert to a previous Kotlin language version of our project within the Project.xml file which then fixed our issue. This is a major limitation as it was outside of our control and we cannot be sure that this issue will not occur again and cause a loss in development time.



```
@@ -4,15 +4,7 @@
        <option name="CODE_STYLE_DEFAULTS" value="KOTLIN_OFFICIAL" />
      </JetCodeStyleSettings>
      <codeStyleSettings language="kotlin">
-       <option name="CALL_PARAMETERS_WRAP" value="5" />
-       <option name="CALL_PARAMETERS_LPAREN_ON_NEXT_LINE" value="true" />
-       <option name="CALL_PARAMETERS_RPAREN_ON_NEXT_LINE" value="true" />
-       <option name="METHOD_PARAMETERS_WRAP" value="5" />
-       <option name="METHOD_PARAMETERS_LPAREN_ON_NEXT_LINE" value="true"
/>
-       <option name="METHOD_PARAMETERS_RPAREN_ON_NEXT_LINE" value="true"
/>
-       <option name="EXTENDS_LIST_WRAP" value="1" />
-       <option name="METHOD_CALL_CHAIN_WRAP" value="1" />
-       <option name="ASSIGNMENT_WRAP" value="1" />
      </codeStyleSettings>
    </code_scheme>
  </component>
```

```
        <option name="CODE_STYLE_DEFAULTS" value="KOTLIN_OFFICIAL" />
      </JetCodeStyleSettings>
      <codeStyleSettings language="kotlin">
+       <option name="CODE_STYLE_DEFAULTS" value="KOTLIN_OFFICIAL" />
      </codeStyleSettings>
    </code_scheme>
  </component>
```

Figure 6.5: Unplanned update found in ”Project.xml”

6.3.2 Kotlin

Limitations

The Kotlin language is not a difficult language to write code in but the lack of references to solutions outside of the main documentation was a hindrance to our project development.

The code itself has been written in a way that is user friendly, with Java code being able to be converted in Kotlin instantly but if you want to look for code that is unconventional and outside of the norm for a class in either Java or Kotlin, you will most likely struggle to find any source to put you on the right path.

The best you can hope for in a Java variation of the same code and convert it before modifying it to suit what you need, This type of limitation will be rendered a non-issue in the future, as the language becomes more popular.

6.3.3 API Calling

API calling was a major feature of our Application throughout our project as it allowed us to generate ratings for each hour shown and returns back a recommendation on which day to best play on.

Throughout our implementation of this feature we came across several limitations and issues within different aspects of the project from the data retrieved to the API calls themselves and even limitations with pricing.

Below will focus on the many issues and limitations that we came across when trying to implement this feature.

Issues

One of the first Issues we had before using the Accuweather API was when we were experimenting with multiple weather APIs, the first of which was the Met Eireann API[15]. This was a fantastic API that would've given us all the data we needed and because it was designed for the island of Ireland, it would've been extremely accurate for the given locations but the issue with this API was its data it returned.

The data returned was in XML format not JSON format and the technology that we were using to fetch and store the data required JSON data. We initially attempted to try to convert this data from XML to JSON but since we weren't storing this data to a file locally it wasn't possible to convert it to JSON. As a result, we had to search for another API to use.

The next major issue when trying to find an API to use for our project was when we tried to use an API called Openweather API[16], it had ticked nearly all the boxes for us, it was capable of finding the correct locations, retrieved back data in JSON format but unfortunately, when attempting to store the data it wasn't able to be correctly stored in our data files as it came back as a set of JSON objects, not JSON data which would allow the data to be temporarily stored so that the data could be processed and return an output to a user. After testing these APIs we eventually came across the AccuWeather API which allowed us to retrieve, store and process as we saw fit.

Limitations

As we were working with these APIs we came across many limitations that wouldn't be called issues but limited our design and focus of the application to try and incorporate them into our application.

When attempting to work with the Open Weather API, one of the very few limitations that we had was that we had to use a third party to try and retrieve the location that the user was searching for based on either longitude and latitude or searching by name, we originally planned it to be searched for by name but found that when searching by name, due to Ireland having unusual city, town and in our case golf course names, it was rather difficult to search via name for some golf courses and because of this, we had to remove the idea of allowing the user to search by name.

We then attempted to search by longitude and latitude for the user using the Geolocation API by google but found that when had we used it, it only returned back the users current location which hindered us from finding the locations of the golf courses we needed. Instead, we gave them a list of golf courses in Galway that when any one of them is clicked, it passes through the location code used by Accuweather to search for the golf course area.

After trying many different APIs we eventually settled on the Accuweather API as it did just about everything we needed to complete the development with our project but came with just one limitation that was very similar to the issue found in the OpenWeather API was when we tried to fetch the five-day forecast, we could retrieve it with no issue but when trying to store it, our data could not handle JSON objects and instead threw errors, with this we had to redesign our what the user was able to see when generating the ratings and with that, we decided to check for the current conditions so that the user could accurately check the current conditions for their designated course as they got closer to the golf course.[14]

The biggest limitation we had with the Accuweather and Openweather APIs was the limitation around pricing. As we are students we could not afford to purchase API packages to get more API requests per month and to have more API fetch types like searching for the weather for the next 72 hours instead of 12 hours and because of this, we had to design the app to be checked the day you plan to play golf to check for the best hours to play instead of being able to plan a couple of days ahead.[14][16]

This also hindered our development as we could only make 50 API fetch requests a day and found when trying to test this functionality after only an hour and a half into development we were unable to continue testing until the following day.

6.3.4 MongoDB

With the way, we used MongoDB with our application we needed to have a MongoDB Realm service app. Realm was only bought by MongoDB in April 2019 for $39 million and had slowly been integrated with the MonogDB services during the past 2 years. [6] The integrated system was only released for General Use on February 4th, 2021. [7] We had been using this system since October 2020. We were allowed access to this due to already having a MongoDB account and can be seen as an experienced MongoDB NoSQL Database User.

Issues

One of the lingering issues we had with connecting to MongoDB from a Kotlin Android Application was that MongoDB 4.4 has been in beta since June 2020. [8] This is the only version of MongoDB that can use with MongoDB Realm Sync. This issue has led to some aspects of MongoDB not being documented fully and integrated to the Kotlin language with query searches being limited. We wanted to only output 5 of the 7 fields we had stored in our MongoDB Database but due to the integration of Kotlin code not being finished, we could only remove 1 of the fields. We hope that in the future this issue can be resolved and our application can be modified to allow the user to choose what fields it wants to generate their data from.

Limitations

With our application, we have a way for the user to create an account to store data within the MongoDB database. At this moment we do not have a way to link a userID to a specific email/password for better user data protection.

This would also render the settings userID as redundant because we could set that value when we log in to the application. This would make all collections that contain the specified value to only open when the email and password that is linked to it is inputted into the application and sent to the MongoDB Realm app to validate their login credentials.

When a user registers an email, due to us not owning a domain that can be used to send an email address verification link, the email address at this time does not get verified. The only real safeguard is when to try to register the same email again, you will be given an alert that states email already has an account.

This can be a hindrance for a user because if you register with an email address that is spelt incorrectly, you will not be told with an alert. The email will automatically be verified by the MongoDB database. The sign-in function with MongoDB Sync was riddled with bugs and is unusable. This led us to remove it from the application entirely until it is fixed.

We hope that when the Realm Sync is fully functioning, that this aspect of the system is working fully and we can use either the function call or purchase a domain for email verification.

6.3.5 Overleaf

For writing our dissertation, we decided to use Overleaf because of our previous knowledge with overleaf from a previous module(Research Methods). The biggest issue we found with overleaf was that you could not have more than 2 people with access to the same project. This led to issues with making sure that every member of the group had access to the latest version of our dissertation. The only solution to this, would be to subscribe to Overleaf with a monthly fee that every member of the group would have to pay.

6.4 Issues Overcame

6.4.1 IntelliJ

With the base version of IntelliJ we could not access a MongoDB database to view the collection data or any changes we had made in the project without leaving IntelliJ and signing in to our MongoDB Project and go to Atlas to view the database collections. This was a big issue with our IDE, as we needed to view all changes instantly so we could find any bugs straight away and patch the problem.

The way we overcame this, was to upgrade our IDE to IntelliJ Ultimate. This allowed the database tool inside Jet Brains to be used in conjunction with IntelliJ. The database tool allows us to sign into our MongoDB database inside of IntelliJ and view all the collections in our database. We could also see any changes made to the database with a refresh tool. This helped us see what was happening quickly and be able to understand what the issue was and how to fix it.

6.4.2 MongoDB

MongoDB Atlas Cluster Integration

The original MongoDB Atlas Cluster region we have designated with our application was not able to run MongoDB 4.4. The use of MongoDB 4.4 was essential to our app because it allowed MongoDB Realm integration. We first had our cluster region set as AWS / Ireland (eu-west-1). This cluster still uses MongoDB 4.2 and could not be used for our application. We had to rebuild our MongoDB Atlas Cluster set up and change the region to AWS N. Virginia (us-east-1). This was the closest region to Ireland that was still a part of the the free tier and was integrated with MongoDB 4.4.[8]

Connecting the Android App to MongoDB

One of the major issues we had with the project was getting access to the Database while the application was running, this led us to find MongoDB Realm. this is a new software to the MongoDB Services. It has been slowly integrated into the MongoDB database since the start of 2020, the general public has only been able to use this since February 2021.

The software created an app on the server side to be used as a gateway between an application and a MongoDB database, The use of this software rapidly changed how we could utilize a database with our application, With only a few lines of code, we could connect to the app to the database and it allowed us run the app offline and when the app connected to the internet again any data stored would be sent to Realm when the user logs in again. The documentation to use it with Kotlin is not finished yet but we could resolve any issues ourselves. [10]

MongoDB returning collections

With MongoDB not enabling a userID to be linked to a specific email and password, we had to find a way to protect user data while printing out data the user has stored in the application. To solve this issue, we created a userID field in the settings(SettingsFragment.kt) page. The user will create a unique identifier in this location that can be used throughout the app. When the user saves a score in the score fragment, the userID is sent along with all the other values. When we enter the User History page of the Android Application, the onCreate function will run a search of the database and bring back every collection associated with the userID that the user has defined on that page.

MongoDB adding Collections

The MongoDB Documentation does not give you the Kotlin code equivalent to adding a collection that is populated with both user input and generated input. To overcome this we had to send all the user inputs to a function that saves the variables into a local variable to be used inside the function itself. The data is then sent to the collection with the insertOne method found in the ScoreFragment.kt file. It used .append for every field to add a new value in the specified field inside a MongoDB Database Collection.

6.5 Future of Application

6.5.1 Multi-platform Application

In our future development plans for our application, we plan to introduce our app to more platforms, we originally planned for our app to be deployed to both android and iOS from the beginning but to focus on android to allow for a more robust application to be designed.

iOS

We hoped to include iOS as part of our plan for development of our app from the beginning but due to iOS being in alpha stability for Kotlin App development[2] we had to remove it from our project. This is due to its unreliability and we had to hold off until the component becomes more reliable so that we can implement it when its status improves.

Harmony OS

When Harmony OS becomes more prevalent on the wider market, we plan to build a version of our app to be deployed to Harmony OS. At the moment Harmony OS only supports Java, C and JavaScript[17] with plans for Kotlin to be supported to later in its development life cycle. When this is implemented and supported, we will then work to add our App to this platform.

6.5.2 MongoDB Realm

Integration

With MongoDB Realm still not fully finished yet, we will see major features added for use with Kotlin. This includes running multiple queries and being able to get back the data in a string, not a document. These two features alone will change the way we get back collections from the database. We would be able to only bring back data that the user wants to see rather than everything except for the ObjectID.

When the login function is fully functional, we will be able to verify email addresses and be able to link a UserID to a specific User. This will improve the security of user data immensely. The email verification would help users who may make a mistake in the email input and may not know what they did wrong.

In the future we would like to be able to link accounts to either your Facebook or WhatsApp to allow the user to share their golf scores with their friends.MongoDB allows users to sign up with your Facebook, Google or Apple accounts. This feature is another that we would like to add to the application to allow the user to sign up in a way that is less time consuming and will not require email verification.

6.5.3 API Call Licensing

For our application to be further improved, we would have to increase the amount of requests the application can make per day as due to being limited to 50 per day, the app is not ready for public use just yet, with the purchase of a higher tier API package from Accuweather we can make at a minimum, 225,000 API calls per month.

Depending on demand for the application, we can then upgrade the package further to meet the demands of the user base and upgrade accordingly to allow for new users to make use of our application as well.

6.5.4 Nationwide/Global

One of the biggest updates to our Application that we wish to introduce, is the inclusion of more golf courses nationwide and even globally, as we want to make this app versatile in all locations and not just limited to one area.

Ireland

Within Ireland we only have locations for 13 golf courses in Galway selected to demonstrate how the application will function. Our future plan is to include not just the rest of the golf courses within Galway but to incorporate every golf course within the island of Ireland. This is to allow the user to be able to search up any course in Ireland and be able to plan golf trips with the best conditions to play from any part of the country even if they are far away from the golf course.

Major Golf Courses

After incorporating all of the golf courses in Ireland to our application, the next big update to the golf courses available to the user is to include many of the Major golf courses around the world so that if you intend to visit one of these golf courses, you can find out which hours or days are best for you and plan your trip around it.

6.5.5 New Features

As the application evolves, we hope to incorporate new features to assist the user during their game, we hope to add features that will also help the user improve their performance and create better golfers.

Wind Direction

One of the features we wish to add is similar to a compass where on their phone the arrow will change in the direction of which the wind is blowing and give its speed as well so that the player will be able to rely on it to adjust their shots direction and club choice.

Club Recommendation

Another feature we wish to add is a recommendation of which club to use on a specific hole, this would be done based upon previous users records to judge which club to use on that particular hole. This will be based upon how successful and popular overall that club was on that particular hole. If a club performs well and used by the vast majority of users, that club would then be recommended to the user.

Weekly Ratings

One of the sections we wish to implement is that of weekly ratings for the next seven rolling days following the search. It was originally part of our design but due to budget limitations we were not able to incorporate this feature. This will allow the user to be able to plan trips a week in advance with an accurate prediction of how the weather will be for their trip.

Chapter 7

Conclusion

7.1 Summary

At the beginning of the college year our team had set their goals at obtaining new knowledge and skills in relevant and interesting areas of application development. Upon agreeing on an idea for an app, The team researched different technologies which were both relevant in industry and new to them. Kotlin ended up being the preferred choice of the team as it has been the preferred android development language since 2019 [3][18].

Upon researching Kotlin more it led to us using the IntelliJ IDE which was the most popular IDE for Kotlin development. Due to Kotlin's multi-platform feature not being stable enough for development, The project became solely Android based.[2]

The team had prior experience with MongoDB which we obtained during our time in G.M.I.T.. We knew that we could use this tool or similar platforms for our applications back-end storage.

When we researched MongoDB we found its new feature Realm which creates an application on the MongoDB server that acts as the gateway between an Atlas cluster and an outside application.

With our app idea being based on golf courses and weather we had to source suitable API's to retrieve store and process appropriate data. We found that we could use the OKHttp request/response API[1] to retrieve weather data from the AccuWeather API. We could not source a golf course API within our budget and as such a hard coded work around had to be implemented.

7.2 Outcomes

- To learn a new programming language
- Utilize a new IDE
- Use an unfamiliar SDK(Software Development Kit)
- To obtain Independent learning skills
- To be able to work as a remote team
- Apply methodology practice in a remote work environment
- Utilise multiple communication tools
- Optimized version control skills
- Achieved successful outcomes using software which haven't been fully released for production environments
- Successfully design and create a full stack application

7.3 Insights

This project gave our team a glimpse into working on a full stack applications development process and the many challenges faced on a regular basis. The team learnt to cooperatively solve problems together and made compromises on their goals due to the unforeseen bugs and limitations encountered which are part of the nature of app development.As a result of creating our Application, we have gained a new found appreciation into the workload it takes to design an application of this nature.

We can take this knowledge into our future careers and be able to apply this to any future projects. The design of the functionality for the application at the beginning of the project is noticeably different to what the team had originally envisioned, but the core features of the application remain the same. The group learnt to adapt and change their approach to match the development tools required throughout the development life-cycle and as such gained skills in solving critical and creative solution based problems.

With the Covid-19 pandemic the team learned to adapt to a remote working environment. While they were used to interpersonal interaction on a daily basis and having access to college facilities such as labs and the library available on campus.

With the team's working environment completely altered by the pandemic, it allowed them gain skills in a remote work environment using tools such as Discord, Microsoft Teams and Intellij's "code with me" feature[19], which allows multiple users to interact with a single IDE work-space at the same time. The development of these skills will be of great value to the team members as they could find themselves working under similar conditions even after the pandemic is over.

The group can also take away the knowledge that they are capable of learning a programming language and how to use an SDK independently and be able to use those skills to develop a fully functioning application.

7.4 Closing

The last nine months have been a difficult year for everyone as the team have had to complete their final year studies under extraordinary circumstances. With this in mind the group rose to the occasion and stuck with a consistent work regiment of meetings and sprints. The team accomplished their goals and delivered an application with all its intended functionality.

We felt we worked really well as a team, everyone supported each other and helped each other out when difficulties arose to make sure the application was to the highest standard that we could produce while staying on schedule.

When developing the App, there was great excitement among the team as we were developing an app everyone had a genuine interest in .This led to everyone having ideas of where they wanted the app to go, resulting in no shortage of concepts for the app, generating an enjoyable atmosphere while creating an app everyone was excited for.

We would like to thank our Supervisor Kevin O'Brien for helping us every week with any issues that may have arose and for his insight into how to structure our Dissertation and Project. His help with our Dissertation was second to none and allowed us to create a Dissertation we can be proud of.

Bibliography

[1] Square, I., 2021. OkHttp. [online] Square.github.io. Available at: ¡https://square.github.io/okhttp/¿.

[2] ”Stability of Kotlin components — Kotlin”, Kotlin Help, 2021. [Online]. Available: https://kotlinlang.org/docs/components-stability.html.

[3] ”Google I/O 2019: Empowering developers to build the best experiences on Android + Play”, Android Developers Blog, 2021. [Online]. Available: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html.

[4] ”MongoDB Clusters”, MongoDB, 2021. [Online]. Available: https://www.mongodb.com/basics/clusters.

[5] ”Announcing MongoDB Realm & The Realm Sync Beta — MongoDB Blog”, MongoDB, 2021. [Online]. Available: https://www.mongodb.com/blog/post/announcing-mongodb-realm-realm-sync-beta.

[6] ”Sharing the MongoDB Realm Roadmap — MongoDB Blog”, MongoDB, 2021. [Online]. Available: https://www.mongodb.com/blog/post/sharing-the-mongodb-realm-roadmap

[7] ”MongoDB Realm Sync is GA — MongoDB Blog”, MongoDB, 2021. [Online]. Available: https://www.mongodb.com/blog/post/announcing-mongodb-realm.

[8] ”Announcing MongoDB 4.4: Available Now in Beta — MongoDB Blog”, MongoDB, 2021. [Online]. Available: https://www.mongodb.com/blog/post/announcing-4-dot-4-available-now-beta.

[9] ”Kotlin docs — Kotlin”, Kotlin Help, 2021. [Online]. Available: https://kotlinlang.org/docs/home.html.

[10] ”MongoDB Realm â€” MongoDB Realm”, Docs.mongodb.com, 2021. [Online]. Available: https://docs.mongodb.com/realm/cloud/.

[11] D. Scott and B. Jones, ”The Impact of Climate Change on Golf Participation in the Greater Toronto Area (GTA): A Case Study”, Journal of Leisure Research, vol. 38, no. 3, pp. 363-380, 2006. Available: https://www.nrpa.org/globalassets/journals/jlr/2006/volume-38/jlr-volume-38-number-3-pp-363-380.pdf.

[12] ”Coolors - The super fast color schemes generator!”, Coolors.co, 2021. [Online]. Available: https://coolors.co/.

[13] ”Flaticon”, Flaticon, 2021. [Online]. Available: https://www.flaticon.com/search?word=golf.

[14] ”AccuWeather APIs — API Reference”, Developer.accuweather.com, 2021. [Online]. Available: https://developer.accuweather.com/apis.

[15] ”Met Ã‰ireann Weather Forecast API - data.gov.ie”, Data.gov.ie, 2021. [Online]. Available: https://data.gov.ie/dataset/met-eireann-weather-forecast-api.

[16] ”Weather API - OpenWeatherMap”, Openweathermap.org, 2021. [Online]. Available: https://openweathermap.org/api.

[17] ”Documentation - HUAWEI HarmonyOS APP”, Developer.harmonyos.com, 2021. [Online]. Available: https://developer.harmonyos.com/en/documentation.

[18] ”Stack Overflow Developer Survey 2020”, Stack Overflow, 2021. [Online]. Available: https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted.

[19] ”Getting started with Code With Me — IntelliJ IDEA”, IntelliJ IDEA Help, 2021. [Online]. Available: https://www.jetbrains.com/help/idea/code-with-me.html.

[20] Kotlin Youtube - How to Quickly Fetch Parse JSON with OkHttp and Gson (Ep 2). 2017. Available: https://www.youtube.com/watch?v=53BsyxwSBJk&t=874s

[21] S. Bose, A. Kundu, M. Mukherjee and M. Banerjee, ”A COMPARATIVE STUDY: JAVA VS KOTLIN PROGRAMMING IN ANDROID APPLICATION DEVELOPMENT”, International Journal of Advanced Research in Computer Science, vol. 9, no. 3, pp. 41-45, 2018. Available: https://pdfs.semanticscholar.org/c0ee/43434064520cdde7222318bf6c4d2db69177.pdf.

[22] M. Flauzino, J. VerÃssimo, R. Terra, E. Cirilo, V. Durelli and R. Durelli, ”Are you still smelling it?”, Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse on - SBCARS ’18, 2018. DOI: 10.1145/3267183.3267186

[23] ”google/gson”, GitHub, 2021. [Online]. Available: https://github.com/google/gson/blob/master/UserGuide.md.

Chapter 8

Appendices

Link to GitHub Repository and Screencast: `https://github.com/stevenJoyce/4thYearGroupProject`