

Fantasy Premier League

Assistant Android

App

**John Grossi**

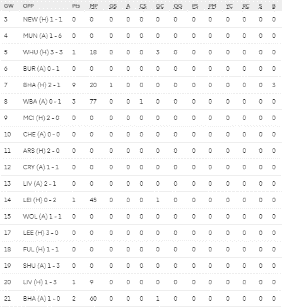
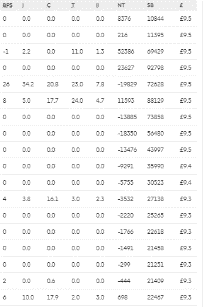
AC40001 Honours Project

BSc (Hons) Computing Science

University of Dundee, 2021

Supervisor: Dr Daniel Rough

***Abstract –*** *Fantasy Premier League (FPL) is a free-to-play game that lets anyone become a fantasy manager at the top level of English football. In the most basic form, managers have a budget to pick players from teams and have to decide who plays each week, given one free transfer each week. It is a game of strategy in which FPL players must predict which football players will get them the most points and have them in their team.*



*The aim of this project was to design and develop an Android app which will benefit players of FPL in making transfer decisions. The app includes features such as predicted line-up predictions for all fixtures, a player comparison tool, and a best FPL team predictor for every week. The official app does not include these, but these features are what players of the game seek, as known by user surveys given out, and they provide a positive impact on their decision making, as known from my research into them. To ensure the app achieves the goal of helping FPL users, participants were involved in the requirements gathering, prototyping stages and at the end of the project to evaluate the app, to ensure all features are ones that the average FPL user is seeking, and that app’s UX is fit for purpose.*

# 1 Introduction

Every week millions of FPL players are left with the same tantalising question: “who do I transfer in this week?”. The main objective of FPL is transferring in and out different players, trying to get the best eleven for that week’s fixtures to score the highest number of points. But how do you know who to bring in? This is decided on several factors: who they play for, who they are playing against, form, numbers of goals, assists, clean sheets (not conceding a goal), if they are fit/available to play and several other factors. FPL gives you all this data and more, as you can see in Figure 1. This is great for the people who have a great interest in stats and numbers; but for the average FPL player, this screen can be daunting. If you want to compare two players, you must click on the player, look through the spreadsheet and then memorise the numbers you want to compare. You then must do the same for the other player you want to compare with. This is highly impractical given that comparing two or more players to decide which one to include is such a crucial element of gameplay. Having such a poor and intimidating user experience can be a turn off to new and current players of the game, and it was felt that this was crucial to address this issue.

Figure 1: Data that is shown when clicking on a player

What was proposed was an Android app that would achieve two objectives. One is to include features that FPL players want, and would benefit from, that the official app/website does not provide, to help make transfer decisions. The second being to improve on the way users compare players they may be trying to decide between, instead of the separate spreadsheet system FPL currently uses.

To achieve this, the following steps were required. Construct research into other FPL assistant app features and construct a survey to ask FPL players directly what features they want and what data they care about. While doing this, research how to implement these features or why they are important or what impact they would have on a player’s team. For example, research prediction algorithms that will produce the best results in fantasy sports or why it is important to know the starting line-up of a team from an FPL perspective. Next, create paper and high-fidelity prototypes to let users give feedback on the features and the overall UI of the app. Lastly, after development, construct user evaluations and compare each feature against a benchmark to record the success of each.

In summary, the primary goal of this project was to help FPL player’s overall user experience with fantasy football, by having access to features that are not available from the official app, and to have a positive impact on their decision making when it comes to making player transfers.

# 2 Related Work

In recent years, FPL has seen a recent surge in popularity when it comes to content outside of the official site. From the many YouTube channels uploading videos about their team selections and tactics, to apps designed to help players make their transfer decisions. To see what other apps are doing and, sometimes more importantly, what they are not doing, some of the most popular FPL Android apps were downloaded.

## 2.1 Other FPL Apps

Five of the most popular FPL Android apps found were: Fantasy Football Fix (100,000+ downloads), Fantasy Manager (50,000+ downloads), Fantasy Football Manager (1,000,000+ downloads), Fantasy Football Assistant (10,000+ downloads) and FFHub (10,000+ downloads). After downloading all of these and trying them, I created a table to highlight their features. A full write up of the apps is in Appendix A. Two of the apps I found did not offer much in terms of new features. They seemed to just be re-skinned versions of the official app. These did not offer much for me as I wanted to gather the full range of different features offered by popular apps to use as prompts in my user survey. The other three apps all offered different features that I decided to delve into further to see how much of an impact they might have on a user’s performance, and the most effective ways to implement them.

## 2.2 Predicted Line-Ups

Knowing who will start in a Premier League game is very important for different reasons. It is not as simple as more playing time means more chance of doing something (although, this is important) – it is due to the scoring system in FPL, as seen in Appendix B. Each player gets one point for playing in a game and a second for playing 60+ minutes.

Now, let us look at goalkeepers and defenders. These types of players are defensive-minded, so their aim in a game is to not concede goals as they get points for not letting it happen and lose points if they do. They only get clean sheet points from playing 60+ minutes and not conceding. So, it is important when picking defenders to make sure they are going to start, so they are more likely to get these points as they cannot get this unless they then come on before the 30th minute of a game. This is highly unlikely as first half substitutes are very rare, and defenders are the least substituted players (next to GKs) [1]. Therefore, it is critical to pick defenders who will start the game.

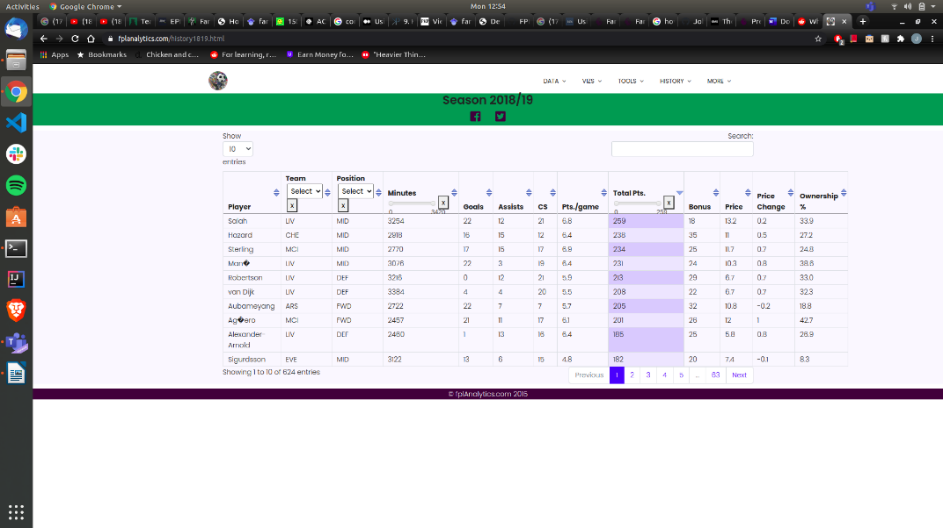
Midfielders and forwards also benefit from starting games rather than coming on as a sub. Both types of players get most of their points from scoring goals. Figure 2 shows players that play a full 90 minutes or even start the game, but then later get subbed off, score more goals than players that get subbed on. So, knowing who starts games can be vital to picking players for your team, from defenders getting clean sheets and, a more likely scenario of, your forward players scoring goals, all of which results in players getting more points.

Figure 2: Table showing number of goals from players depending on time played from the 12/13 PL season [2]

## 2.3 Best FPL Team This Week Prediction

When choosing a structure for the algorithm, I looked at other people’s attempts at predicting the best FPL team and general sports prediction algorithms. After searching I narrowed my focus to two algorithms that both showed promising results and had similar structures to what I wanted to use.

First was a model proposed by Bonomo et al [3]. This model was based on the Argentinian fantasy football (Gran DT), which is “run by a major Argentinian newspaper for the first division of the real Argentinian professional soccer league”. The scoring system and transfer system has the same idea as FPL with a few additions, such as points awarded by the newspaper for man of the match (MOTM). The model works by averaging the past three weeks points for a player to then predict how many points they would get in the upcoming. This number is then multiplied by a number between 1.05 and 0.95 four times depending on the following things: playing at home (x1.05) or away (x0.95), league position (1 to 1.05, if in the bottom five of the table, or 0.95 to 1 if in the top five), and a number between 1.05 and 0.95, depending on a scoring streak. Lastly, this number is then multiplied by 1 or 0 - 1 if they are expected to play and 0 if not. So, if someone has a high expected number of points but isn’t going to start (say, because of injury), their points are set to 0, as they won’t play. There is an exception for if they think the player will be substituted on, the player will get x1 instead of 0.

The second algorithm was a linear/lasso regression approach proposed by William Eilertsen et al [4]. Regression algorithms are common tools in sports prediction, but for the purpose of this review, I am going to be looking at the specific one proposed by William. He splits the model into 3 steps: position, variable selection, and then fits it into a regression model. The variables are the realised points (actual points gained), team, position and home/away. There are then the numerical variables: each week’s previous points, price, transfers in/out, minutes played in each game, yellow/red cards, goals, assists, penalties missed/saved, saves and clean sheets. In the paper they recognise that each position gets different points for different things. They categorise them into each position before continuing with the regression. This is what lasso regression is – by not factoring in, for example, clean sheets for a striker, it gives a more accurate point prediction as it does not matter in a points perspective if a striker’s team does not concede a goal. The model then takes in training data, which is done to determine the accuracy and the value of error. This error is then accounted for.

## 2.4 Why Look at The Stats?

Why not look solely on real-life ability when making transfer decisions? One of the biggest mistakes that new or existing FPL players make is going for “big name” players. When they are given the choice of players, they will go for all the players that are heavily talked about in the media or their social circles. However, this is not the way to look at fantasy football. Fantasy football is a stats game, not a popularity game. Even though pundits will go on about how good some players are, it does not always transfer over to FPL. Two examples of great players not transferring well into FPL are N'Golo Kanté and Virgil van Dijk, with Kanté winning the player of the year in the 16/17 season and Van Dijk winning the same honour in the 18/19 season. Both players had excellent seasons, coming in for big money moves, and were instrumental to their teams doing well that season, with Kanté helping Chelsea win the league and Van Dijk steadying a leaky Liverpool defence.

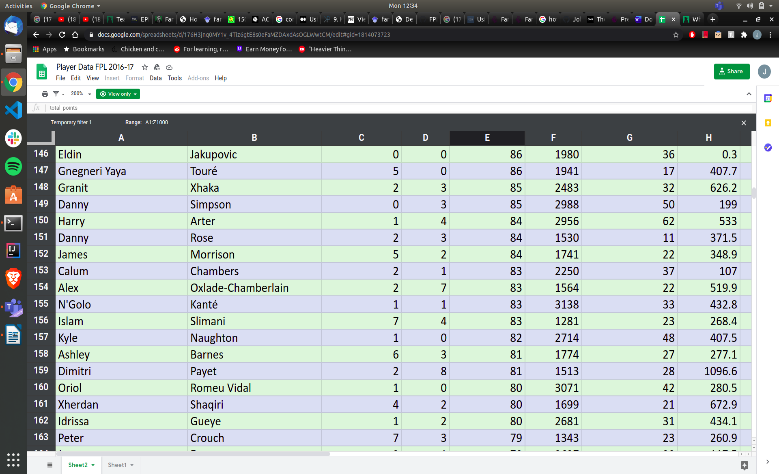


Figure 3: 16/17 FPL player season stats [5]

But, as Figure 3 shows, real-life ability does not always move over to FPL. The season that Kanté won the player of the year award, he only came in 156th place for points. A drastic 181 points off the topmost points. The main reason he is not a good FPL option is due to his role as a player, which is a defensive midfielder. A midfielder will only get one point for clean sheets. Therefore, with this system, Kanté does not get as many points as the defenders who get four points for a clean sheet. Since Kanté does not offer much in the attacking sense of goals and assists, and only gets one point for his defensive contribution, he is not a good FPL option.

Figure 4: 18/19 FPL player seasons stats [6]

Van Dijk suffers a different problem compared to Kanté. Van Dijk is a highly renowned player and he greatly improved Liverpool's defence, which resulted in them winning more games [7]. When he is playing, Liverpool has a 70.5% win rate and concede an average of 0.94 goals per match. When he is not playing, Liverpool’s win rate decreases to 42.9% and concede an average of 1.57 goals per match. Conclusively, without Van Dijk playing Liverpool concede more goals and lose more games. Conceding fewer goals mean more clean sheet points and, if they are winning more games, they must be scoring more goals. These statistics show that Van Dijk is fundamental to Liverpool playing well, keeping clean sheets and is highly praised, winning player of the year that season. However, Van Dijk was still not the highest-scoring defender, or even the highest scoring Liverpool defender for that matter in the season he won player of the year, as seen in Figure 4. Andy Robertson (Liverpool, left-back) has outscored Van Dijk every season for Liverpool, as seen below in Figure 5.

Figure 5 : Robertson vs Van Dijk season stats [FPL website, in transfers]

The main reason for this is because Robertson is a full-back and Van Dijk is a centre-back. Centre-backs are traditionally more defensive and full-backs can be more attacking. Since full-backs have this freedom to go further up the field, they are more likely to get assists more often, where centre-backs are more likely to get goals from corners (due to their height) but far less often. In summary, a player should not be picked solely based on their on-pitch ability, but instead how likely they are to do things that result in gaining points and not losing points.

This stat-based approach is most commonly known as a ‘Moneyball’ approach. Instead of using scouts who use a subjective approach, Billie Bean (an American baseball coach) used a stat-based approach when bringing in new players with their team’s small budget. This approach was later written into a book by Michael Lewis called ‘Moneyball: The Art of Winning an Unfair Game’ [8], and has been shown to work in football, as Liverpool have adopted this approach with much success [9].

**3 Requirements**

**3.1 Gathering Requirements**

To gather my requirements/features, the best way was to go straight to the people who would use the app and ask them what they want. The survey included questions to find what features they would most like to see in an app, what data they use to make decisions and what other medias they used to help them make transfer decisions. The survey was created using JISC surveys[[1]](#footnote-1).

The first few questions were things such as how long they had been playing FPL, what other apps/websites they use for FPL and if they look up team news before games. This was to get an idea of how much experience users had with FPL features and the UX. Asking users what other apps and websites they used was to see if they used any at all because, if the majority did not, then it could be argued that there is not anything out there that people feel is worth using. If they had used other medians, I wanted to know which ones so I could see the features that they provided. Lastly, the reasoning for asking if they looked up who was predicted to start was to see if this was a feature they would use. In previous research it was shown that if a player starts, they are more likely to get points, so this was a feature that was thought about early on in development.

The next question was a list of features from the research into other FPL apps and the research into which features could have a positive impact on a player’s transfers. This was to see what features users thought were most important or useful. This could help narrow down the features that could be used in the application. Each option prompted the user to rate it from ‘not at all useful’ to ‘very useful’. The two following questions were if a paywall or having to provide your email address would make the person less likely to use the app. This was to identify what things may stop users using the application before they even get to see/use the features.

The final four questions were asking what data they look at when looking to get a player for a position. So, for example, “When choosing a DEF transfer which of the following aspects do you look at?”. This was then followed by a list of stats they could choose from, and then the survey prompted the user to add any other stats they look at that were not mentioned. This was to gain knowledge as to which stats people care about and think are important when making their transfer decisions between players.

**3.2 Survey Results**

The survey was answered by twenty-four FPL players. The full survey and responses can be found in Appendix C. Thanks to this, I was able to see what feature people were most passionate about and what data they wanted to be shown to compare players to make their transfer decisions.

The survey results showed some interesting results, such as that a large percentage of users look up who will be starting games and are not put off by entering their email but are by a paywall. Some of the users expressed their biggest problems, such as knowing which players are in good form, who to pick as their captain and struggling to pick between two players to bring into their teams.

The next set of results to be looked at were the features that users thought would be most useful. The most unanimous result was the ‘this week’s fixtures’ feature with seventy-five percent of users saying it would be very useful. Having a starting line-up predictor, player comparison tool, ‘best FPL team this week’ predictor and a feature showing the user their live rank proved to be met with a positive reaction. All the other options were either not positively received or were mixed. Lastly, all the data that people said they looked at when making transfers for each position were recorded.

**3.3 Final System Requirements**

With the aim of the project being to create an Android app, which will benefit players of FPL in making transfer decisions, the features had to be decided by looking at the features that people wanted and which features would provide points. The first feature to be decided on was the fixtures feature. As previously mentioned, it was the most popular feature in the survey for what feature people would want. Next was the feature that predicted the starting elevens of each fixture. This was chosen as it was a popular choice in the survey, but also that in previous research, it showed that if a player starts a game, they are more likely to do something that results in getting points. The final two features that were picked were a player comparison toll and a best team for the upcoming week predictor. These were chosen as they were again popular in the survey. The player comparison also was a perfect fit for the aim of the project as, when making transfer decisions, many of the times you will want to compare the player you have in your team with another to decide who is best to have. As a result, the final fixtures that were decided upon were as follows:

- The upcoming week’s fixtures

- The predicted elevens of each fixture

- A player comparison tool

- Best team for the upcoming week predictor

**4 Project Management**

**4.1 Methodology**

When deciding on a methodology to use as my approach, I looked at the positives and negatives of some to decide which would be a best fit for the project. After much consideration, a mostly Agile approach was decided upon. One of the reasons was because I had expected many obstacles to arise during the software development due to many reasons. I had never created a mobile application; it had been some time since I had programmed in Java, and I had not had a great experience programming and managing a back-end of an application. Since an Agile approach allows for flexibility and quick restructuring, which a Waterfall approach does not, it made the most sense to go with this approach. After my requirements were gathered, I turned them into user stories and held all of them in my sprint backlog. These could then be split up into individual sprints which would focus on different features. This is where I then decided to make use of a hybrid approach. Getting a review from FPL players at the end of every sprint was going to be difficult as I would need them to be available every week. Therefore, I got user feedback multiple times before the software development, consisting of a survey, paper prototype focus group and a high-fidelity prototype focus group, followed by a final user evaluation after development. In addition to this formative and summative feedback, my advisor and I held a weekly meeting on Microsoft Teams for around an hour to discuss what I had done that week and what I would be doing in the upcoming week. This helped me keep motivated and was useful to get regular feedback on how I was progressing and what I could improve on.

**4.2 Tools & Technologies**

Throughout the whole project many tools and technologies have been utilised.

GitHub[[2]](#footnote-2) was vital to the development of the project, as it was where I could hold all my files and have version control for all of them if something were to break while editing them. Being able to split all the different features into different branches helped separate these features into encapsulated, easily managed sub-tasks. After a feature was created and tested, I knew I could then merge it into the main branch and have no problems. Some other alternatives were briefly considered, such as GitLab and BitBucket, but GitHub was what I had the most experience with and I knew it was fit for purpose to manage the iterative inclusion of new features with version control.

When it came to creating my Android application, I chose the Android Studio IDE[[3]](#footnote-3). Again, I had no previous experience making mobile applications, so I had to research what would be the best approach. Android Studio is the official IDE for Android and has all the tools I would need to make an Android app. It has a large, active support community, and no other IDE came close to the tools and convenience that Android Studio offered. Eclipse is the only other IDE that appears to be a working alternative, but it requires the installation of a plugin to enable Android development. Also, Android Studio offers an easy drag-and-drop feature to easily edit the xml files, which Eclipse did not have, as well as an emulator to see the application running. Having an Android phone helped with development, as Android studio made it easy to just connect my phone by cable and have it run with no problems.

With version control and my Android IDE chosen, I then had to choose how I would approach the back-end of the application. I wanted the back-end to be responsible for the resource-heavy operations to avoid the app being a large size to download, and reduce loading times as much as possible. So, I wanted the back-end to: get the data I needed, store the data, and manipulate it before the app requests it. For example, every time somebody requests the best team for the upcoming week, I want it to be in the database already instead of having to get every player every time and having to run an algorithm to decide who should be getting displayed to the user. After much consideration, I decided to use Firebase[[4]](#footnote-4).

Firebase is a backend service which can be used for mobile and web applications. They provide NoSQL databases, cloud functions and an array of other tools. Firebase documentation is extremely extensive and is supplemented by a YouTube channel of helpful tutorial videos which made it very appealing to use. Also, being owned by Google, Firebase was integrated into Android Studio. So, getting it set up and connected with the app was made much simpler. The cloud functions could be used to take off much pressure from the front-end by being able to time them to run at specific times on specific days to perform tasks. For example, getting team news from external sites or running my prediction algorithm were implemented as timed cloud functions. So, the front-end would just need to get the players from the database and then display them. Another advantage was the amount of free storage, reads, and writes Firebase offered, which could not be rivalled by many other services. Because of all these features previously mentioned, Firebase was decided upon. Another alternative was a XMPP server, but compared to Firebase, its documentation was not as extensive. It is more difficult to set-up and would be more difficult to connect to my app than Firebase. The last option would be to create my own server to host my back-end, but due to time constraints imposed by the task deadlines planned in the project schedule [discussed in Section 4.5], this didn’t seem to be the best course of action.

Next, a tool to manage my sprints and product backlog needed to be chosen. ClickUp[[5]](#footnote-5) was exactly what I was looking for. It has a template that made it easy to make a product backlog, where I put all my user stories and sprints. It has useful features for viewing all user stories and which sprint they are in, what stage they are at in development, story points and an easy-to-use interface. There are many other project management tools I could have used here which would have achieved the same goals. but ClickUp was decided on simply as it was intuitive to pick up, so I did not need to spend time learning how to use another and had all the tools I wanted to help manage all my user stories for each sprint.

5 https://clickUp.com

For the prototyping, there were two types: paper and high-fidelity. For the paper prototypes, no software was going to be used; but after some consideration, it was decided to use a piece of software called Marvel6. This allowed the drawn paper prototypes to be put into the software where it could then me made more interactive and easier to share with other people. The software made it possible for drawn buttons to work like actual buttons, in the sense that if one was clicked it would bring up the corresponding screen that would come up in the final product.

For the high-fidelity prototypes, Adobe XD7 was decided upon. The advantage of using this was familiarity with the software from previous projects, an easy-to-use interface, the ability to create responsive UI, and to have the ability to share with other people to get constructive feedback.

**4.3 Diary and Meeting Minutes**

Throughout the whole project, notes were taken to keep track of what was getting done, and to have a better idea if deadlines were going to be met. Daily notes were taken (see Appendix D) until the software development started, as my sprints were keeping a record of task progress done that week. Meeting minutes (see Appendix E) were kept with my advisor to keep a record of what was discussed every week. This helped to be able to look back on and make sure everything that had been mentioned in the meetings was being addressed that week.

**4.4 Backlog/Sprints**

Before the software development could begin, a product backlog and sprints had to be planned out. First, all the user stories were added to the backlog, which was created on ClickUp, and can be seen in Appendix F. After this, each user story was assigned story points to estimate how long each task would take. After this, it was planned that after each sprint the next sprint would be filled with tasks, which resulted in the story points adding up to the length of the sprint. If the tasks were finished before the sprint was finished, more tasks were added, and if they were not finished, they would be added to the next sprint. Each sprint was assigned to be one weeklong, and for there to be a total of eight, initially, with there being time for more to be added, if necessary.

**4.5 Gantt Chart**

6 https://marvelapp.com/

7 https://www.adobe.com/uk/products/xd.html

At the start of the whole project, a Gantt chart was created to estimate the roadmap of the project. In Appendix G, you can see the whole project plan. This included each process of the project, with estimates of how long each would take. Milestones were added, with dates that deliverables would need to be ready by to be submitted.

**4.6 Risks**

With an Agile methodology approach being used, this allowed for a flexible approach to the project. With change being expected, it could be managed better than, for example, if a waterfall approach were used. This allowed for if some features took longer than expected or if illness hindered the progress of the project, then this time could be made up for.

One of the biggest worries I had is with COVID-19 still being such a prominent part of everyday life and, with multiple national lockdowns, the English Premier League could have been stopped. The government had stated that, despite the new lockdown, elite sports may still continue behind closed doors [10]. However, some games were occasionally getting called off, and I had to be prepared that the league could have been suspended. If this were to happen, I had planned to pick a date and create the app from that point, as there will not be any more games to be played. Pick game week five, for example. I would have had the data before that week and would still have a list of the games after it before the season had been cancelled. Provided the league did not get called off very close to the submission deadline, it was expected that there would be time to adapt the app.

With the FPL API having no documentation, it was a risk that it might have not be possible to get all the info needed from it. If this were to happen, it was planned to use web-scraping to get this data elsewhere.

**5 Design**

**5.1 Paper Prototypes**

After consulting my background research and my user surveys, I decided on the following features:

* This week’s fixtures
* Line-up predictor for each fixture
* Player comparison tool
* Highest scoring team predictor

The background research helped me realise that knowing who starts games often results in more points, so this is important for people to know if they want to improve their scores. A player comparison tool was one of the most popular results from the survey and is not a feature offered by the official app, or even many other apps currently on the market. Finally, prediction algorithms of the best FPL team that week have great results, according to my research into them. This was a popular option in the survey, and not many other FPL assistant apps offered this. Overall, all my features had a combination of a positive reception from the survey and from my background research, but also was offering something that that other apps on the Play Store were not.

With these features in mind, I created a paper prototype to encompass them, which can be seen in Appendix H. These were then put into Marvel, which lets you turn your paper prototypes into more interactive ones, where clicking the buttons drawn on the prototypes will move you to the next screen. This was sent to three FPL users, and a focus group was constructed. The interviewees were asked what they liked and did not like about the prototype. The array of features and clear design were the favourites of the focus group, but some things, for example the predicted line up not being clear that it was not the official one, were brought up as a negative. Feedback was recorded in Appendix I and was applied to my high-fidelity prototype.

**5.2 High-Fidelity Adobe XD Prototypes**

All the feedback from the paper prototype was taken into consideration and applied to the production of the high-fidelity prototype. This was created using Adobe XD, and can be seen in Appendix J. This was also then presented to a focus group of the same three people. The interviewees, again, were asked what they liked and did not like about the prototype. The group really liked the inclusion of the player images being used, but found that some of the buttons, for example the budget on or off button, was too small and not very clear. Feedback was, again, recorded in Appendix K and was used when creating the final product.

**6 Implementation**

When it came to my software development cycle, as previously mentioned, it was decided to take it on with a mostly Agile approach. There were originally only eight sprints, but by the end, there was a total of ten. This was expected as I knew things would take longer than expected, as I was unexperienced in many things, such as the language, tools, and environments. Each sprint was a weeklong, lasting from Monday to Sunday, and each had a focus of a different feature.

**6.1 Sprint 1 14/01 - 17/01**

The first sprint was a bit shorter than the preceding sprints. This sprint lasted four days, and the main focus was getting myself familiar with the technologies I was using and setting myself up for the rest of the software development. During this sprint, I was able to set up and become familiar with Android Studio and get an app to move between three screens, which would hold my three main features. After this, I wanted to set up my back-end and become familiar with how it would work. I created a Firestore database on Firebase. This is a NoSQL database that is flexible for mobile development. It keeps data in sync across apps, but also offers offline support, so the app is still responsive regardless of internet connectivity. When this was set up, I was then able to connect my Android Studio project to the back-end (Firestore). Being all owned by Google made this fairly simple. Now that all of this had been done, I spent the rest of this sprint becoming familiar with the FPL API. This is the official API but has no documentation at all. After some research, a Medium article was found [11], which contained all the endpoints and the data it would retrieve. All these endpoints were recorded for future use.

**6.2 Sprint 2 18/01 – 24/01**

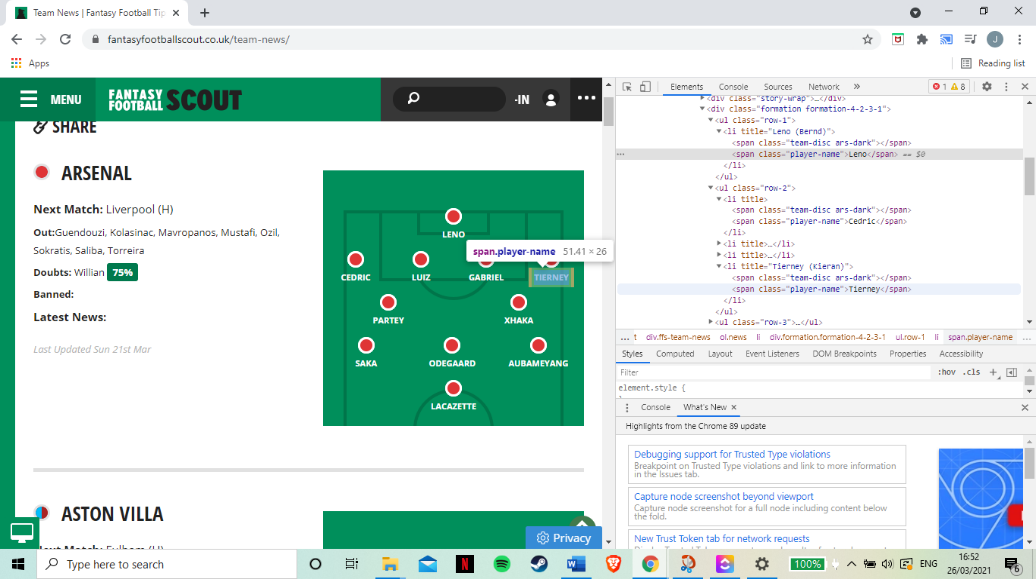
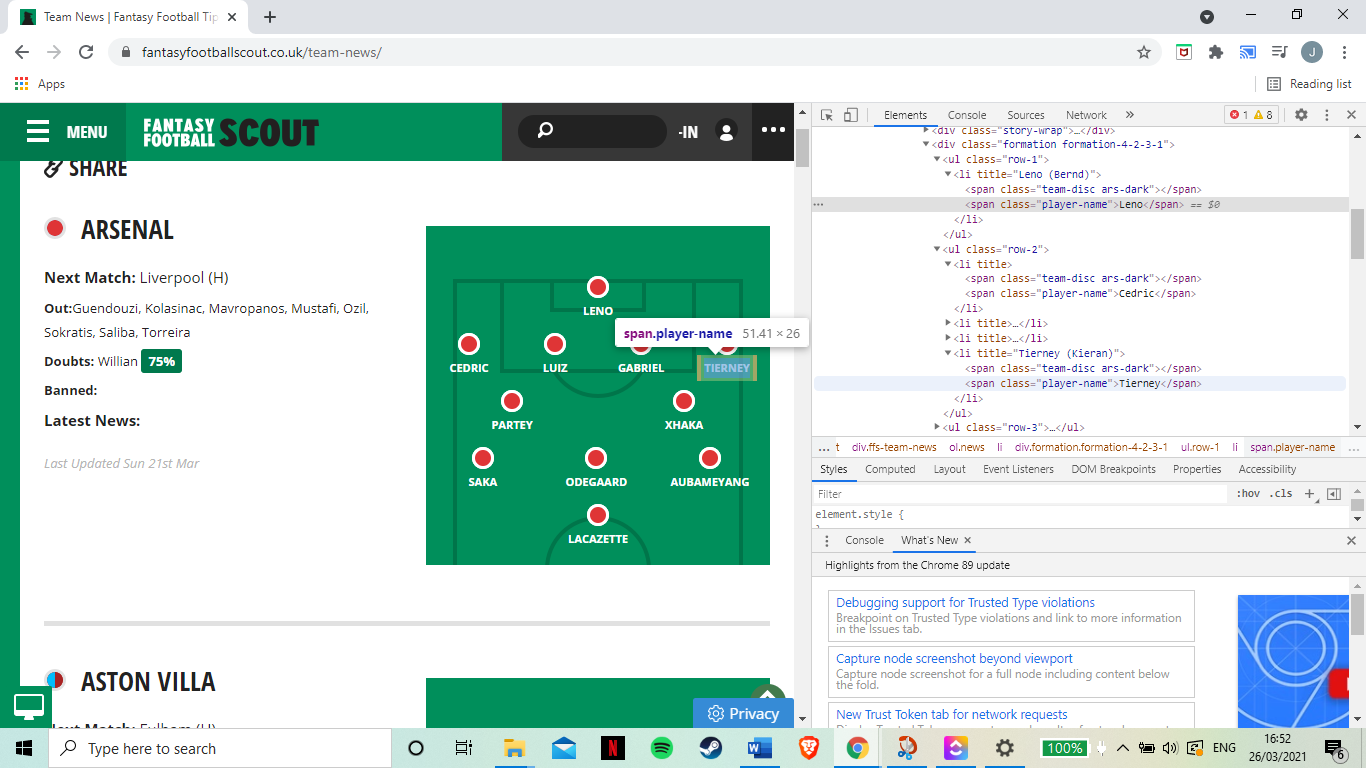
Sprint two was the first full sprint, and the focus was on the predicted line-up feature. The idea with this function was that a user would click on an upcoming fixture and it would show the user the two teams predicted elevens of players who will play that fixture. Firebase offers cloud functions, which can be ran at certain times. So, with these, the idea was to write a cloud/Firebase function that would predict the eleven players for each team and then write them to the database. The function worked by going to three different websites and web-scrape the eleven players for each team.

Figure 6 : FPL Scout predicted 11 page, showing HTML

Each site required a different approach to scrape their data but for an example Figure 6 shows one of the sites and the HTML of the page. What the scraper would do is then shown in the figure bellow, Figure 7.



Figure 7 : FPL Scout web-scraper code

The function requests the webpage that contains all the team line-ups and loads the HTML. With the HTML, it then goes through each player name under the “.player-name” tag, gets their last name and then adds it to a player array. Once it has done this, it then sets a JavaScript object to have a nested object, which is the first team’s name. This object is then set to hold the eleven players predicted to start. This loops until all twenty teams have been added with their predicted elevens (E.g., teamANDpalyers {Arsenal { 1:Leno, 2:Holding ….} Aston Villa {1.Martinez…). This nested JavaScript object is then returned to the method, which is responsible for then comparing this object with the two others retrieved from the other two sites. The code works on a 2:1 ratio. So, if two sites think, for example, Mendy will start in goal for Chelsea, and another thinks Caballero will start, the code will add Mendy, since that was the majority. This is done for all the positions for each team, which results in an object with the same structure as the one mentioned previously. This object holds the twenty teams, with each team object holding eleven players, with these player being the ones predicted most likely to start and were at least 2:1 with the websites.

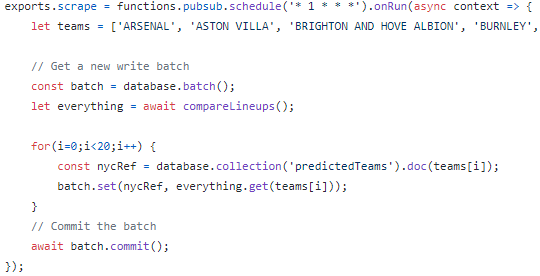


Figure 8 : Code example writing predicted eleven to the Firestore database

This data is then written to the database (see Figure 8) with a path of ‘predictedTeams/TeamName’, where predictedTeams is the collection, TeamName is the document and then all the player names are set to the documents fields.

**6.3 Sprint 3 25/01 – 31/01**

Sprint three was the first sprint that a roadblock was hit. The purpose of sprint three was to do the back-end of the fixtures feature. This feature would show the user the week’s upcoming fixtures, which they could then click on to see the predicted starting elevens of the two teams. The plan was to use the FPL API, as one of the endpoints was to do with all the game’s fixtures. Unfortunately, the end point seemed to be broken. For the first attempt, a simple request was done, but no matter how it was tried the endpoint would not return a readable response. When this did not work, multiple GitHub repo/libraries were tried. These libraries also did not work, so it was concluded that the end point was broken as neither me nor any external libraries were unable to get a response.

Since the API would not work, another approach was attempted – web scraping. I was going to use the same approach as last time, using Cheerio to get the HTML, but it was noticed early that this was not possible. This was due to the page using JavaScript to load the fixtures; so when Cheerio loaded the HTML, the JavaScript would not run to show the fixtures. So, to get around this problem, Puppeteer was used instead. Puppeteer simulates a Chrome window, instead of just getting the HTML. This lets the JavaScript run and the teams then are in the HTML code. Normally this is not available if using an external server, as you cannot make them open Chrome tabs; but with Firestore, they have something called ‘Headless Chrome’, which allows you to do this with Puppeteer. Once the page was loaded, the teams were found in the HTML under a tag and put into an array, where I could then build the fixtures in another array, which is returned at the end of the function. This can be seen below in Figure 9.

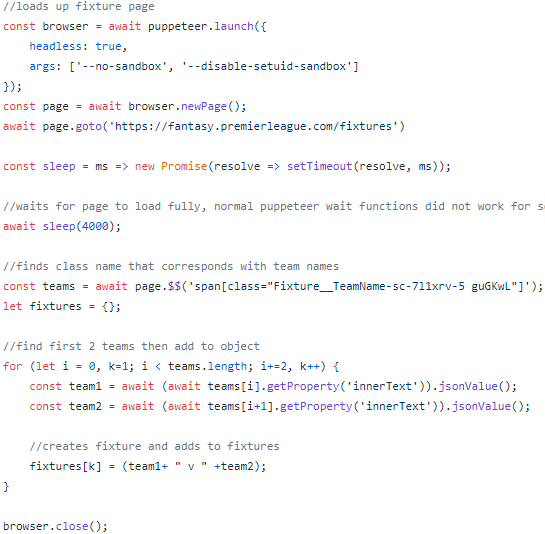
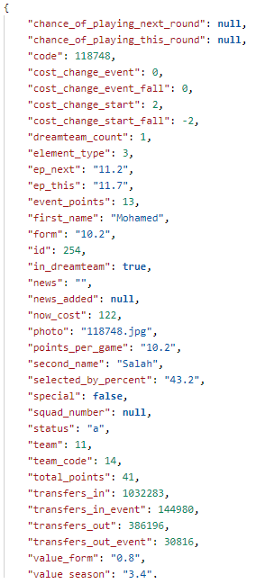


Figure 9 : Web-scraper code

**6.4 Sprint 4 01/02 – 07/02**

Sprint four was a far more productive sprint, with there being less problems encountered, and far more work was able to be done in this sprint than expected. The story points I gave each task in this sprint did not take as long as expected, so I was able to get two features done in this sprint, opposed to just one, being the front-end for the fixtures and predicted line-ups fragments.

With the back-end all set up for these features, all that had to be done on the front-end was the UI, making database queries and presenting the data in the desired way. I set up my database in a way that may have not utilized normalisation, but instead, to make my queries as simple as possible. This was recommended in the Firebase documentation [12].

To display the fixtures, a RecyclerView was used. This allowed the fixtures to be displayed dynamically, as it only made it as large as the number of fixtures received. Each fixture was set to a button title and, when it was pressed, it brought the user to a new activity, which shows the home and away teams’ predicted line-ups. When the activity is loaded up, the two teams from the fixture, which was clicked on, is passed through using intents. The team names they are then used to query the database to get their predicted line-ups. In Figure 10, you can see the database. To query it, the path ‘predictedTeams/team-name/’ is used, where the team names from the fixture is just passed in to get the home and away team players. After this, a tab layout was made to switch between the home and away fragments and a back button was added, so users could go back to the fixtures fragment.

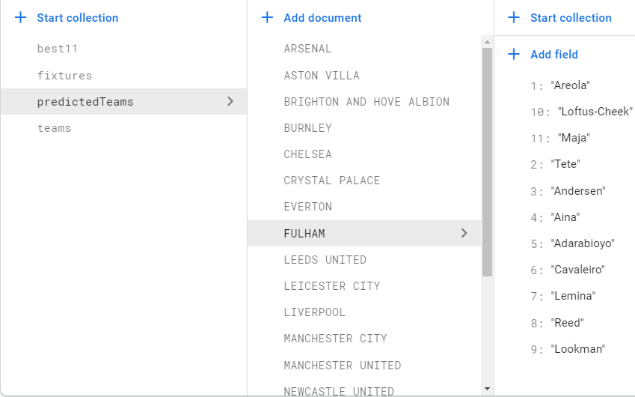


Figure 10 : Firestore database. Example of path to Fulham predicted eleven

From my user feedback on my high-fidelity prototypes, it was brought up that the screen would look better to have the player images above their names, instead of just having a circle. To avoid having the app being larger than necessary, the player images were saved to Firebase storage, where the images could be requested when needed. To get these, a Python script was written to scrape the English Premier League player website [13], where it would load a Chrome window and keep scrolling to the bottom to load all the players. Once it had done this, all the hyperlinks to the player pages were saved to an array, where it then looped through them to get the players images from each link. A problem was noticed here, as the images were saved as the players name .png; but when there were repeats of names, they would just override each other. For example, there are multiple players called Fernandez who play for different teams. To avoid this the images were then saved as ‘playernameTEAMNAME’, so the Fernandez from Manchester United would not override the Fernandez from Newcastle. These images were then requested from storage and displayed above each player’s names.

**6.5 Sprint 5 08/02 – 14/02**

Sprint fives’s main focus was to develop the back-end for the player comparison screen. There were multiple things to be done in this sprint, such as getting the data from the FPL API, manipulating the data into a more readable or beneficial form, and populating the database. Thankfully, the FPL API endpoints for player data was working, so I was able to request the players data I needed. For gathering the data, the first thing that was done was requesting the Bootstrap endpoint that returned every player’s basic data, as seen in Figure 11.

Figure 11 : Salah’s season stats, JSON format

Once this was requested, the function would then iterate through each player that was received through this request. First, the second name is set to just be the last word, as a player can have many names. It sets their second name to be whatever the last one is, as footballers are always referred to by their last names. This was due to consistency across the app and, for when the app requests the image for the player, it has to match the image name. For example, João Pedro Cavaco Cancelo. The second name from the API is Pedro Cavaco Cancelo; but in the predicted line-ups and images, his second name is just Cancelo. So, all the words before Cancelo must be removed. After this, all accents are removed for the same reasons, as the image names do not have accents. This had to be done across many parts of the project. In the predicted eleven function, sometimes it would not get eleven, as some sites would use accents and others would not. This meant when comparing, it sometimes would not work as one site had Ag**ü**ero and another had Ag**u**ero. So, when these names were compared it would not say it was a match, meaning he would not be added to the prediction. So, to avoid this when all the names were scraped, all accents were removed so the error would not occur. Now that that had all been done, the elementSummary endpoint could be used. This requires the players ID which can be found in the Bootstrap endpoint return (E.g., ‘elementSummary/ID’). This returns all the player’s data on how they did in each previous game week and who they still have to play. Lots of the data collected at this stage had to be changed to be more readable. In the API, teams and player positions are always referred to by an ID, so these had to be changed. Score lines are saved as two variables, being the home team score and the away team score, so these had to be written clearer, and the code needed to figure out if the player plays for the home or away team. All of this data was saved to an object with the same structure, as the predicted eleven object mentioned previously, then set to the database. The team data is also set here, this includes the team strength, home and away attack, and the home and away defence.

**6.6 Sprint 6 15/02 – 21/02**

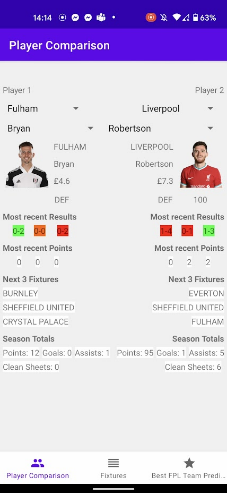
Sprint six’s focus was to develop the front-end for the player comparison screen. Having the experience of getting data from the database in the previous two features made it faster to implement at this stage. The biggest challenge was filtering the data. The player comparison screen was split in half for the two players to be seen side-by-side, as you can see in Figure 12.

Figure 12: Early development of player comparison screen

Code was written to change the colour of their results to show if they won or lost, with green being a win, orange being a draw, and red being a loss. The most problematic part of this feature was definitely the drop-down filters (spinners on Android Studio). The first drop-down would give the user the option to filter by team. Picking Fulham, for example, the dropdown below would then be populated by all the Fulham players. Clicking a Fulham player (Bryan, for example) then displays all the player’s data to that side of the screen, as seen in Figure 12. All the data on the screen is the most popular results from my user survey of what data they looked at for each position when making a transfer, but also what data results in more points for their position. For example, it makes sense to show clean sheets for defenders, as that is their main income for points; but forward players get very little or no points for keeping clean sheets. Different positions show different stats, but everything down to ‘season totals’, in Figure 12, is the same for every position. Goalkeepers have total points, clean sheets and penalty saves; defenders have total points, goals, assists and clean sheets; and midfielders and forwards have total points, goals, and assists.

The code works as follows: the first dropdown displays the teams, and when the user selects one, the code makes a database call with the team name in the path. This returns all the players for that team, and when the user selects a player, it then makes another database call, this time with the team and the player’s name, returning all their stats. Making all four dropdowns (two for each player) work was troublesome. Instead of having four almost identical methods for each, I was able to make it work in one onClick method, using two if statements, with the first being for the team spinner and the second being for the player name spinner. This meant the code had to get the spinner’s ID of which had been clicked, and then change the relevant text views so the data was shown on the correct side.

When the user selects the player, the code grabs the player’s data and then gets the player’s image from Firebase storage. However, every player does not have an image, this is normally due to them joining the team late and not having an official image on the FPL site. So, if this were to be the case, or there was an error requesting the image, a default image would then be used as a placeholder, so the user would not wait for the player image to appear, as that player did not have one. This was also implemented into the other feature screens, and can be seen in Figure 13.

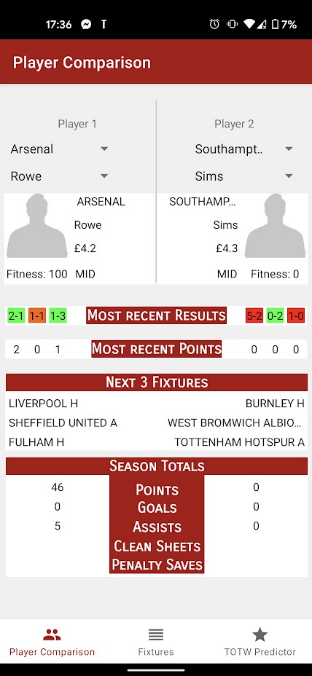


Figure 13 : Default image placeholder if no image is found

**6.7 Sprint 7 22/02 – 28/02**

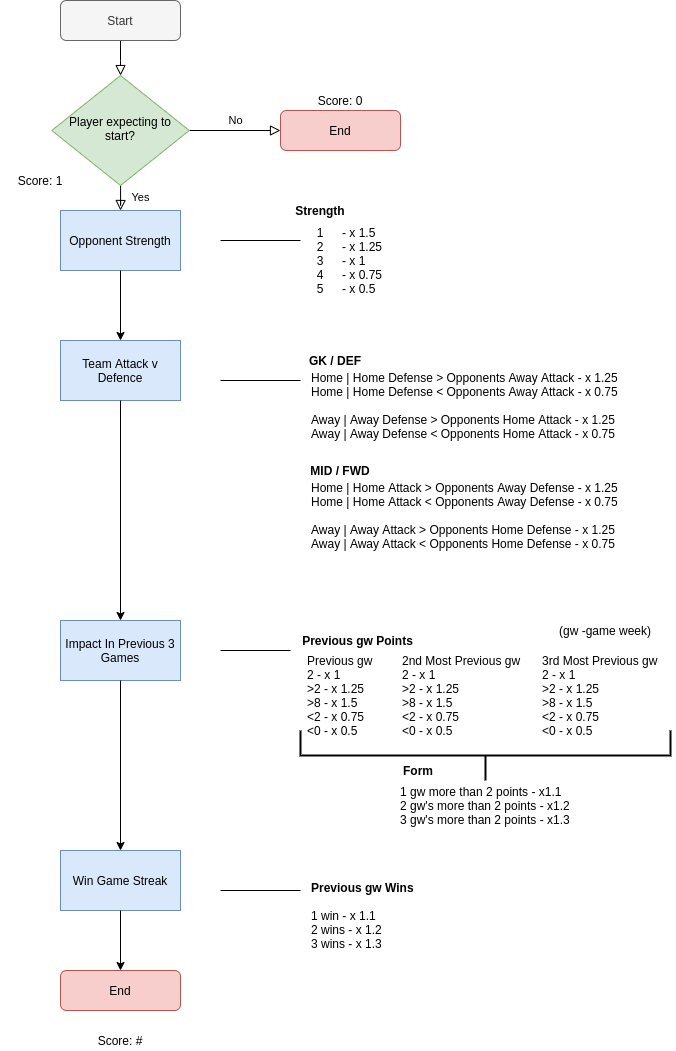
Sprint seven was going to be the start of the development of the best team that week prediction algorithm. The algorithm would be responsible for going through every player and giving them a score depending on several factors, such as past results and the likelihood of them doing well in the upcoming week. Unfortunately, I was unwell this week and not as much work as expected was able to get done this sprint. Fortunately, due to the amount I had managed to get done in the previous weeks and the fact I had accounted for things going wrong, this did not hamper development greatly as it could have. What did get done this sprint was the plan for how the algorithm would get developed, as seen in Figure 14.

Figure 14: Flow chart to visualise how the prediction algorithm works

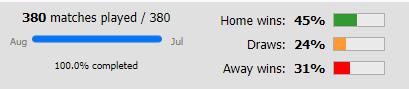
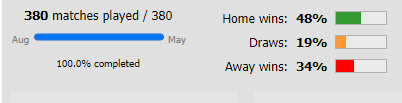
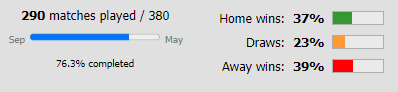
The first step of the algorithm was to check if the player is expected to start. Due to the background research mentioned previously, it was important to know who starts, as this normally reflects in expected points. If they were not expected to start, they were given a score of zero and then the algorithm would move to the next player. If they were expected to start, they were given a starting score of one. Depending on a number of factors, after this, the score would be multiplied depending on their stats, their teams stats and who they were coming up against. This was a hybrid of the two algorithms from my research in section 2.2. Taking inspiration from Bonomo’s idea of looking at past stats to predict the future and having the system of multiplying the players score depending on the stat. Then also taking inspiration from William’s technique of looking more at areas like players positions and home/away dependencies to be more specific of where the points are coming from. For example, in Figure 14, ‘Team Attack v Defence’ looks at the players position and if they’re home or away, and then compares it against the opposite. For example, a forward at home is compared against the oppositions defence away. It was chosen to not solely look at home and away advantage, as that has not been as prominent this year. With no crowds at the grounds since Covid-19 the crowd has less of a chance at inspiring the team when at home, resulting in the home advantage argument not being relevant. This can be seen in Figure 15, with the 18/19 season being the last full season with crowds, 19/20 having crowds for half the season and then finally the 20/21 having none whatsoever.  

Figure 15 : EPL 18/19, 19/20, 20/21 season stats [14]

So, instead of giving a team a positive multiply if they are away, it was instead decided to look at a team’s defence/attack at home/away vs an opponent’s attack/defence at away/home.

**6.8 Sprint 8 01/03 – 07/03**

With sprint seven being responsible for the planning of the algorithm, sprint eight was where the development began. Following my flowchart (see Figure 14), the first step was to check if the player was starting. This was able to be done quickly, as I did not need to iterate through every player. All that had to be done was get the predicted eleven from the database. These player’s scores were set to one. Next, was to get the data relating to them (I.e., past data, future games, etc.). With this, the code could then give the player its score. Each square in the flow chart was made into its own function that would return a multiplier, which is then used to give the player their final score, as seen in Figure 16.

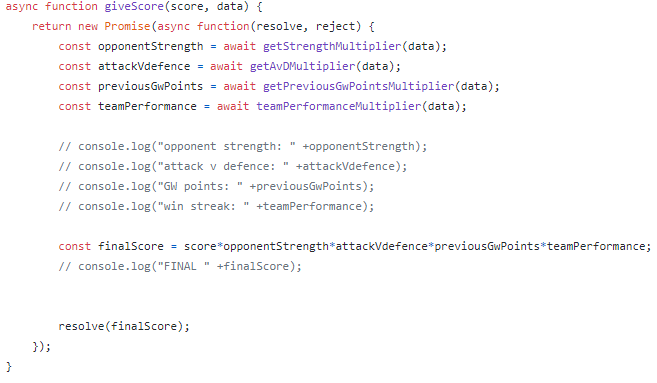


Figure 16 : Give player score code

**6.8.1 Strength**

The first multiplier function, as seen Figure 17, was based off of the difficulty of the opponent. Each team has an overall strength ranging from one to five, with one being the easiest and five being the most difficult. If a team is going up against a stronger opponent, the multiplier is smaller, as they are less likely to do well and vice-versa.

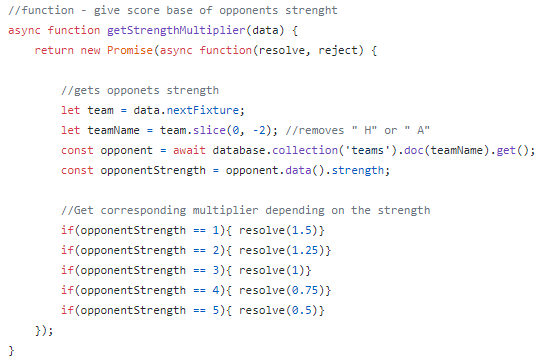


Figure 17 : Code snippet of getStrengthMultiplier function

**6.8.2 Team Attack v Defence**

The second multiplier function, as seen in Figure 18, looked at if the team is home or away, the position of the player, and then the opposite of the opponent. For example, if the team is home and the player is a defender, the code will compare the teams home defence strength against the opponents attack away from home.

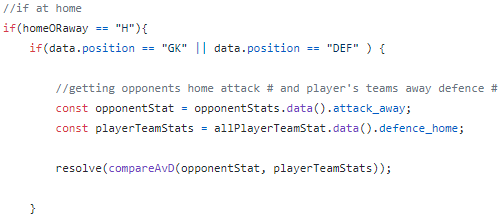


Figure 18 : Code snippet of part of the getAvDMultiplier function

The two numbers are compared in the ‘compareAvD’ function, as seen in Figure 19. Depending on which is larger, the appropriate multiplier is returned.

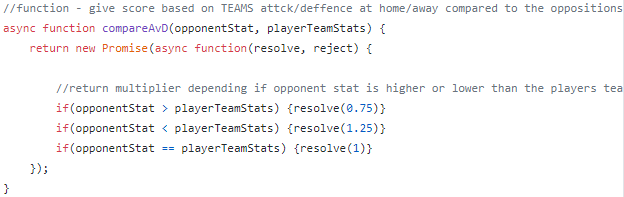


Figure 19 : Code snippet of the compareAvD function

**6.8.3 Player Form**

The third multiplier function, as seen in Figure 20, looked at a player’s form in the past three games, and returned a multiplier, depending on how well they had been playing. The code looks at the points gained in each game and assigns a multiplier. While doing this, the code also records if the player is getting more than two points in any of the weeks, which increases the form multiplier by 0.1 with each game they get more than two points.

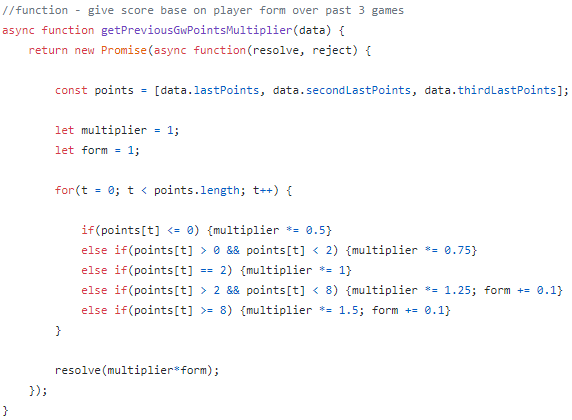
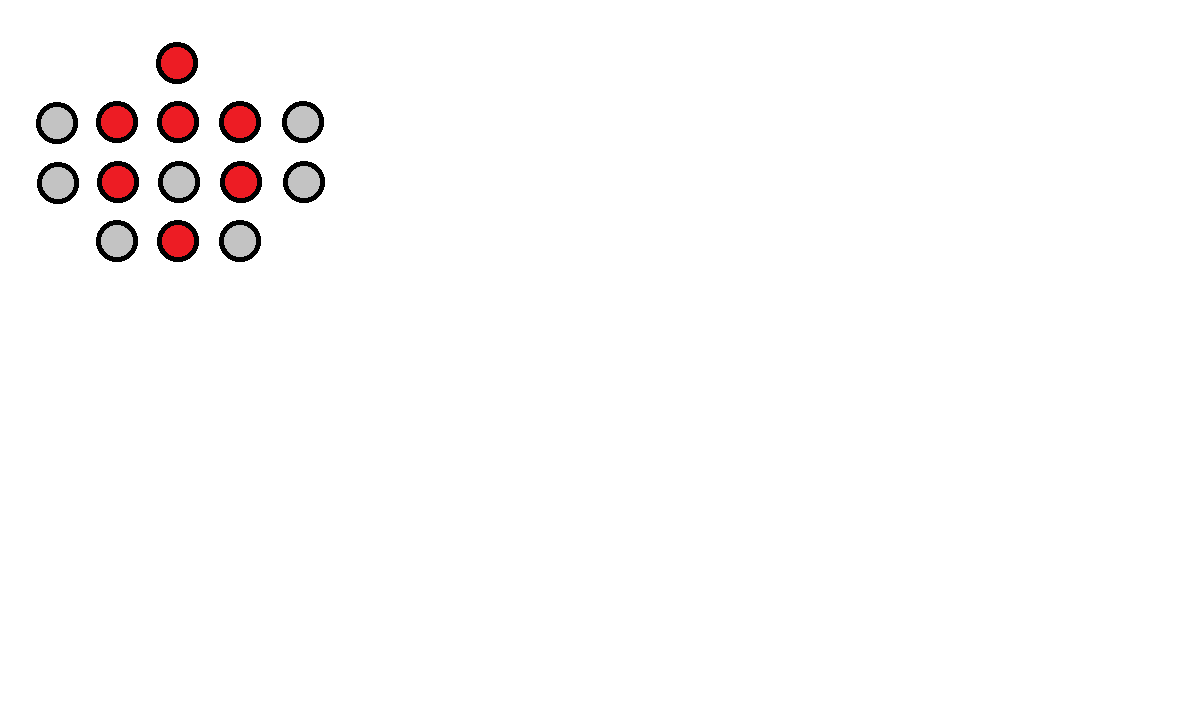


Figure 20 : Code snippet of the getPreviousGwPointsMultiplier function

**6.8.4 Win Game Streak**

The final function, as seen in Figure 21, is responsible for looking at the overall team performance. It looks at what kind of form the team has played at in the last three games. With each win, the multiplier increases by 0.1.

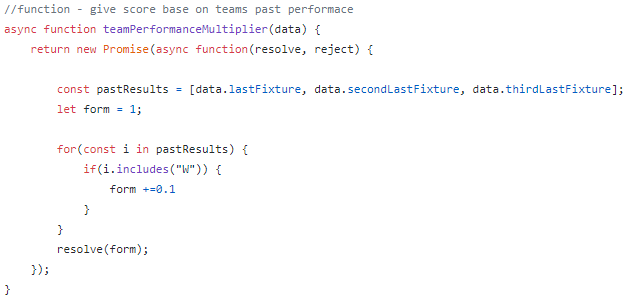


Figure 21 : Code snippet of the teamPerformanceMultiplier function

**6.9 Sprint 9 08/03 – 14/03**

Sprint nine was responsible for solving my constraint satisfaction problem. More specifically, it is called a Multidimensional Knapsack Problem (MKP). The idea of the Knapsack Problem is that you have a bag, and you need to fill it up with the combination that would hold the most value, but not go over the weight limit of the bag. The MKP is the same, but with more constraints than just weight (sizes, for example, could be considered). So, in my situation, the bag is my eleven player slots, and the constraints are as follows:

1. There must be eleven players.

2. There must be at least one goalkeeper, three defenders, two midfielders and one forward.

3. There must not be more than one goalkeeper, five defenders, five midfielders and three forwards.

4. There can be no more than three players from each Premier League team.

So, it is not as simple as just grabbing the top eleven players, as that would not satisfy the constraints. One idea was to get the top three players from every team, which means it would not go above the three player limit, and then make the eleven from that picking the best player for each position. The problem with this approach is that you are not guaranteed to get a player for every position. If no team has a goalkeeper in their top three, then it does not satisfy constraint two.

Figure 22: Valid formation. Red – minimum, Grey – available

The next approach was to work my way down the formation to fill up the minimum formation, as seen in Figure 22, and then fill the rest of the spots with the next highest scoring players or skipping them if there were already too many players from that team. This would satisfy the first four constraints, but there is still a problem. This does not guarantee the highest scoring combination. A simple example would be to say the first three positions (goalkeeper and the first two defenders) were all Chelsea players, with a score of one, two and three. The code would then not let any more Chelsea players in the team, matching the constraints. The problem is if when it gets to the forward position it is possible the highest scoring player could be a Chelsea player, with a score of four, meaning the highest scoring team would not be made. After much consideration of other methods an algorithm had to be made to solve this problem of picking the starting eleven.

Below, in Figure 23, is a diagram of how the problem was solved to meet all the constraints and get the highest scoring predicted team. The first goal is to fill up the minimum number of positions, seen in Figure 17, but in order of highest scorer first, instead of going down each position in order of goalkeeper to forward. As the code gets the highest scorer from the nested object, it removes it, so it does not just get found every time. The array that holds the best eleven is then iterated through, to check if it holds too many from the position or the team. If there is too many from that team, the player is just deleted. If there is too many for the minimum position already it is put into a buffer array. The reason to keeping these players is because, if there is too many for the position at the moment when the minimum is met for every position, the player could then be added to fill up the remaining spots. Although, if there is already three from that team that is never going to change, so it would be redundant to keep the player somewhere, as there is no situation where they will be used. So, once the minimum is met for each position, the buffer array is put back into the nested object holding the players. Then the same process is repeated, but now each position cannot go past its maximum and will stop once it hits a total of eleven players. This way it is always a valid formation and is the combination that results in the highest scoring team, solving the Knapsack Problem I had encountered.

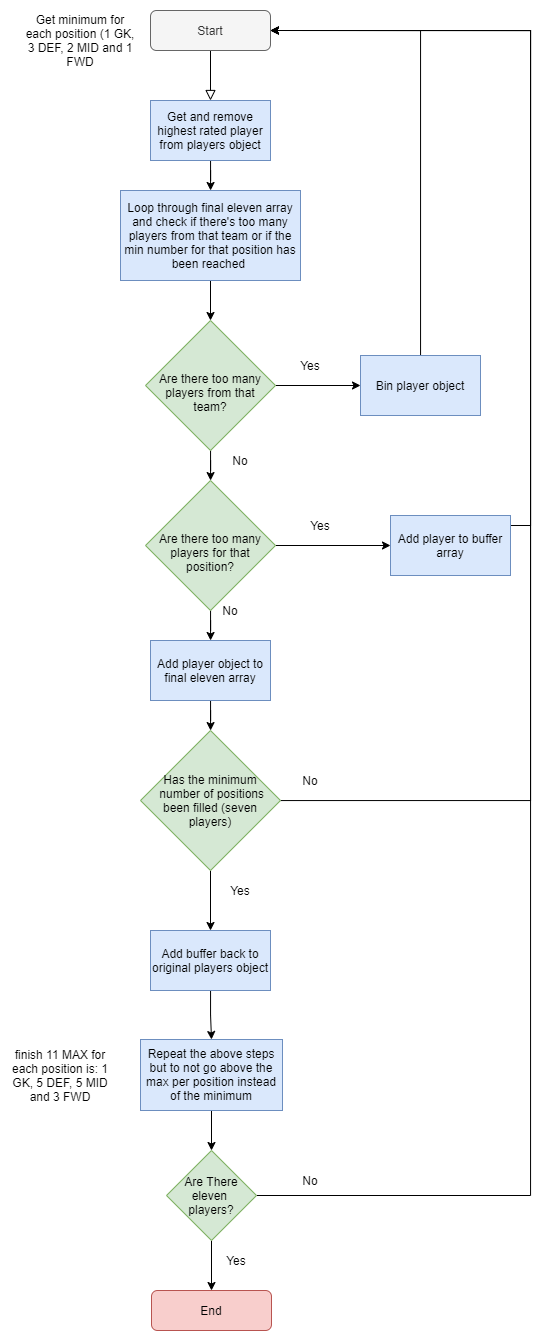


Figure 23 : Flow chart to visualise constrain satisfaction problem solution

This step of the software development was far more difficult than previously expected. It was thought that the algorithm to give each player their score would be more difficult to achieve than getting the eleven players. In reality, it was the opposite. With so much research beforehand, the algorithm was easy to code, as I was well informed and prepared on how it would work. But, as stated, getting the eleven, matching all the constraints, and getting the best combination of players proved challenging, but was achieved. The full code for solving the MKP can be found in Appendix L**.**

Once this was achieved, the front end was developed very quickly, as it was very simple to implement. The best eleven had been written to the database, so a simple database call was needed to get the players. Then, similar to the other screens, all that had to be done was displaying their names with their corresponding pictures.

Whenever anything was written to the database, it had to be done at certain times. For example, player data had to be collected every day to be the most up-to-date. Then, predicted eleven had to be added before predicting the best team that week to have the most recent news on who is most likely to start. All the functions can be seen in Figure 24, and show the time they are deployed, memory allocated to each function, and the timeout times set to each one. These had to be altered to make sure the functions ran smoothly and did not timeout for the ones that had to be given time to run to completion.

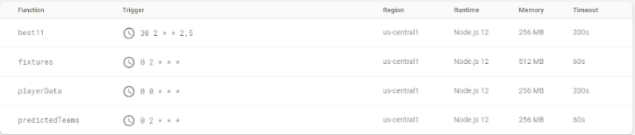


Figure 24 : Cloud function console

These functions ran using Cron time where a calculator was used to check and confirm the times the functions would be ran at were correct. [15]

**6.10 Sprint 10 15/03 – 21/03**

Sprint ten was the final sprint and was dedicated to improving the UI and fixing bugs I had noted throughout development. During this time, I had also performed a final user evaluation focus group and also a few informal user tests. This was just letting a few friends and family try out the app. All of this resulted in some bugs being found and UI improvements that were then implemented in this sprint. Using ClickUp was useful during development and here. It was useful, as below the backlog (in ClickUp) I was able to keep a note of any bugs noticed during development to ensure none were unattended.

**7 Evaluation**

After the software development was completed, the evaluation began. This, again, is where I had deviated from the Agile approach as, normally, an evaluation was done at the end of every sprint; but instead, it was done at the end of the whole development, so I could test the features against others to see how successful they are.

**7.1 Focus group**

**7.1.1 Participants**

The participants who took part in the final system evaluation were the same participants that took part in the high-fidelity prototype focus group. This was done on purpose as it was thought that it would be good to let the participants who saw the high-fidelity prototypes see the final product to see if it matched their expectations.

**7.1.2 Method**

As previously mentioned, during sprint ten a focus group was constructed to review the final product. The Android APK was sent to all the participants the day before, after all the appropriate forms had been read and signed. If the APK did not work for whatever reason, for some of the participants a live demo was prepared to show them how it worked and what they could do in the app. And if they wanted to try it they, could take control of the screen and try it themselves on the emulator. The APK worked for everyone, so this was not necessary. Questions were prepared prior to the online meeting and each participant was cooperative. All the feedback was summarised, put into one page, and shown to the participants to confirm it was the overall thoughts of the group.

**7.1.3 Results**

This resulted in much positive feedback towards the app, such as the overall look of the app and the array of features but the participants also responded with constructive feedback such as bugs they had noticed and things mentioned in the high-fidelity prototypes that had not been added to the app. One of the things brought up was that the app did not work with no internet connection. It showed the fixtures and the player names but was unstable and would crash sometimes. Another thing brought up was that I had not added a title saying when the line-ups had been updated to the predicted line up screen when the players were updated, which I had mentioned in my high-fidelity prototypes. The focus group members were all friends of mine and all knew each other well. This made the meeting run smoothly as everyone was comfortable with each other and did not need to be prompted to contribute, as everyone knew each other, so they were not nervous to speak. They were also more comfortable to criticise the product, as they knew me and did not need to try and be polite to me with their criticism. Although, I am aware since I knew the participants that this would add a level of bias, as it could be argued that since they knew me, they would be more positive with their reviews. Overall, the focus group provided much information into what had worked and things to consider for my future work.

**7.2 Starting Eleven Prediction**

To evaluate my starting eleven predictor, it was thought best to compare them with external sources. I could not use the websites I scraped as I had used their predictions to create mine. So, two others were used. First was TheStatZone, [16] – a very popular site that provides statistics on many sports as well as football. Second was FPLHUB [17] – probably the most well-known and well-resourced FPL platform aimed at helping FPL users. Another reason as to why the evaluation was done after development was that FPLHUB is a paid service and provides a weeklong free trial. So, to test my app against theirs, I had to record the results of my prediction and then get the free trial once I had enough data to compare them. For three weeks the starting elevens of my app were compared to FPLHUB and TheStatZone’s predictions. This can be seen in Appendix M. The prediction never dropped below eighty-five percent, which was seen as a success, with it also beating the other two sites. The way this percentage was calculated was that every right prediction for each team was added up. This was then divided by two hundred and twenty (eleven players for each team, multiplied by the twenty teams) and then multiplied by one hundred to give a percentage.

**7.3 Best Team This Week Algorithm**

To evaluate my algorithm, the team selected was recorded every week for three weeks and compared against the average score, with the average score being of all FPL players. If the algorithm could pick a team that scored over the average, it was taken as a success. The results can also be seen in Appendix M.

The first week the algorithm scored thirty-seven points, and the average was thirty five. This was a positive start, if only by two points above the average. This highlighted the unpredictability that is ever present in football that I cannot account for. One player got injured in the first five minutes of their match, one player did not play as they contracted COVID-19 and Burnley provided a shock result against Everton.

In week two the algorithm provided fifty-two points which is sixteen points more than the average – forty-six. This was a far greater result than the previous week and showed the algorithm was having a promising start by not being below the average. Again, problems arose as one of the players, Rüdiger of Chelsea, did not start for disciplinary reasons, having had a fight in training. This resulted in him not playing and getting zero points.

Week three was the last week and saw the algorithm gain thirty-one points, with the average being thirty-three. This was the third week in a row that the algorithm had beat the average score of other FPL players. Although, like previous weeks, problems were noticed. One being one of the players missing a penalty and, again, a player not being selected because of injury. With all three weeks having a player not play, the idea of having a bench is something to be explored in future work.

Throughout the three weeks the algorithm never beat my personal team’s score, showing that the algorithm definitely had areas that it could be improved on. Looking at more past data is something that could be looked into to improve this. The biggest problem would be to make the algorithm smarter by implementing machine learning. Knowing things like Jamie Vardy having a better scoring record against top level teams compared to lower level teams [18] is an anomaly as most strikers find it easier to score against the lower level teams in the league, as this normally means going up against worst defenders.

**8 Description of the Final Product**

When it came to the final product, every milestone and target for the application were met, bar one. The only inclusion is a budget constraint on the best team of the week prediction feature. This was due to the complexity of trying to solve the MDK problem and trying to get the best combination of players with a budget. The method used by many is to use a linear programming model or a simplex method. These were attempted by looking into libraries and external tools, such as Google OR tools, which unfortunately was not able to be implemented in time.

The first feature to be implemented was the upcoming fixture tab, as seen in Figure 25. This would show the user the week’s upcoming games with the option to scroll through them and select one to see the expected line-ups for each home and away team.

Figure 25 : Fixture screen

The second feature to be completed was the predicted line-ups for the home and away teams, as seen in Figure 26. This shows the home and away teams predicted line-ups, and states at the top of the screen that this is updated every Tuesday and Friday. The fragment utilises two tabs, which you can either click on or swipe to change over to. When showing the players, if there is no image for the player or an error requesting it, the app will display a default silhouette of a player to take its place.

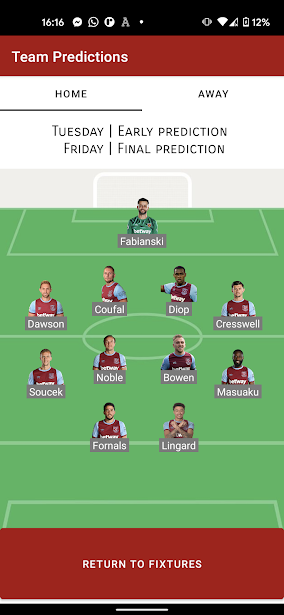
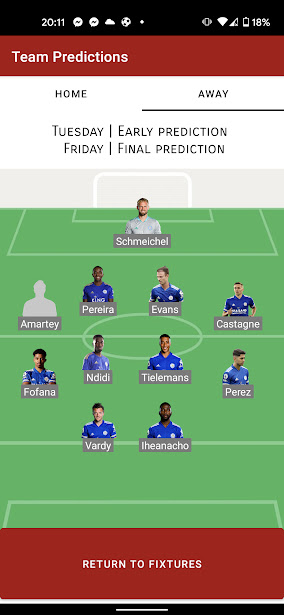
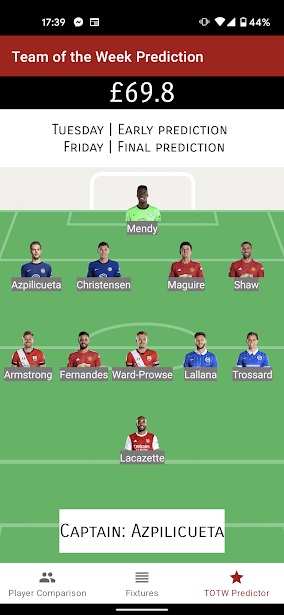
 

Figure 26 : Predicted eleven screen

After this the next feature was the player comparison screen, as seen in Figure 27. This gives the user the option to compare any two players side-by-side. The user can filter it first by the team, and then by the player with the stats shown depending on the player’s position. Past results are colour-coded to signal if they won (green), lost (red) or drew (orange) and their upcoming fixture have ‘(H)’ or ‘(A)’ at the end to signal if the game is being played at home or away for the player’s team.

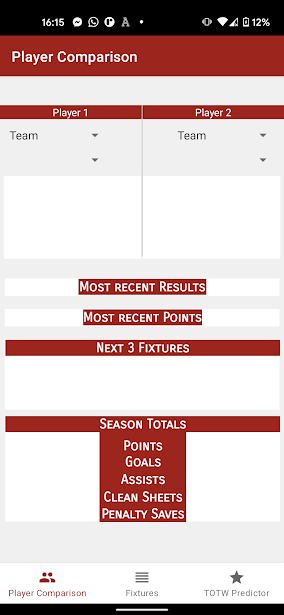
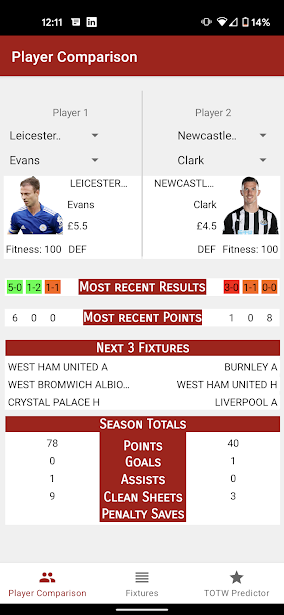
 

Figure 27 : Player comparison screen before and after players have been chosen

Finally, the last feature to be done was the best team of the week predictor, as seen in Figure 28. This feature was for users to see the team that the algorithm predicted to get the most points in the upcoming game week. At the top of the screen the price of the team is shown, and then the text saying when the team is updated. This and the predicted line-ups are updated on these days for two reasons. The first being that most game weeks run from Friday to Monday. This means players may want to make their transfers right after this, so the first prediction is on Tuesday. Although, on occasion, there are midweek games that sometimes start either Tuesday or Wednesday, so updating the database on Tuesday morning makes sure these games are covered. The reason for updating on the Friday is because the closer to the game means a better idea who will start. This is due to the manager press conferences for the games take place late in the week, so player’s fitness and availability is made clearer at this time. Lastly, a captain is picked based on the player that had the highest score from the prediction algorithm.

Figure 28 : Best team of the week predictor screen

The last thing to be added was a simple splash screen, as seen in Figure 29, that is shown for three seconds when the app is started before going to the fixtures page. This utilised the animation feature from Android Studio, where the logo I created could be shown moving down the screen when the app is started.

Figure 29 : Splash screen

**9 Conclusion / Future Work**

The aim of this project was to design and develop an Android app that will benefit players of FPL in making transfer decisions. The motivation of this was that many FPL players have trouble making transfers every week, and the FPL UX for making these decisions was something that I felt could have been improved upon. To achieve this, requirement gathering was conducted to see what problems people faced and what they would like to solve these problems. With these requirements, the paper prototypes were created and, with the feedback of these, the high-fidelity prototypes could then be developed. App development proceeded this for ten weeks, and a final user evaluation was completed shortly after. At the end of this project, I felt like this has been achieved. The user evaluation at the end of development also shared this thought. My feature that predicts the starting eleven of each team has an average of 85.5% success record for the three weeks it was tested and outperformed two other well-regarded sources. Lastly, the best team of the week predictor beat the average score every week.

When it comes to future work, it is not as simple as adding more features. Quantity over quality would hamper and crowd the app. Instead, an iterative design process would be used to improve the features already present. User feedback on the app would be used to determine this.

As for any other future work, the first thing to be done would be to get the budget constraint working on the best team of the week predictor. This is so the user could toggle the budget on and off to see the two variants of the team. Last would be to add a bench to this feature, too. What was noticed during my evaluation was that, often, a player would not play for an array of unpredictable reasons, so it made sense that, instead of just selecting eleven players, it should pick fifteen. This would match what regular FPL players would do as they get one hundred million to pick fifteen players comprised of eleven playing and a goalkeeper, plus three others on the bench. So, when one of the starting eleven do not start, a bench player would take their place.

One of the things that would have to be considered, from a legal perspective, is the images used. The app would not be ready to be put on the Android Play Store, as the player images used are copyrighted images. What would be used instead would ne similar to strategy to FantasyFootballFix [19], a fantasy football assistant website. As seen in Figure 30, they use player drawings to take the place of the image of the player. 

Figure 30 : FantasyFootballFix player images

Lastly, the English Premier League fixtures are protected by copyright. Football DataCo [20] is the company that allows other companies outside of the English Premier League to list the fixtures. So, to put the app on the Android Play Store, a fee of £266 would need to be paid.

**Acknowledgments**

I would like to extend my thanks to Dr Daniel Rough for the constant guidance and support given throughout the project. Also, a great thanks has to be given to anyone who took part in the requirements gathering or evaluation of any part of the development. The feedback and results were vital the development of the system.

**References**

[1] - Gomez, M., Lago-Peñas, C. and Owen, L., 2016. The influence of substitutions on elite soccer teams’ performance. International Journal of Performance Analysis in Sport, 16(2), pp.553-568.

[2] - StatsBomb. 2021. Smart Use Of Substitutes Can Make A Difference | Statsbomb. [online] Available at: <https://statsbomb.com/2014/01/smart-use-of-substitutes-can-make-a-difference/>

[3] - Bonomo, F., Durán, G. and Marenco, J., 2013. Mathematical programming as a tool for virtual soccer coaches: a case study of a fantasy sport game. International Transactions in Operational Research, 21(3), pp.399-414.

[4] - William Eilertsen, Akash Gupta and Bjørn Kåre Kristiansen, 2018, Developing a Forecast-Based Optimization Model for Fantasy Premier League, pp.39-44.

[5] - Google Docs. 2017. Player Data FPL 2016-17. [online] Available at: https://docs.google.com/spreadsheets/d/176H3jnq0MY1v\_4Tiz6gtE8s0eFaMZDAxdAsOGLWwtCM/edit#gid=18 14073723

[6] - Fplanalytics.com. 2019. Season 2018/19 | Fplanalytics. [online] Available at: <https://www.fplanalytics.com/history1819.html>

[7] - Planet Football. 2020. Comparing Liverpool’s Record With And Without Virgil Van Dijk Since 2018 - Planet Football. [online] Available at: https://www.planetfootball.com/quick-reads/comparing-liverpools-record-with-andwithout-virgil-van-dijk-since-2018/

[8] - Lewis, M., 2013. Moneyball. New York: W.W. Norton.

[9] - Goal.com. 2020. 'It Is Brilliantly Executed' - Moneyball Guru Praises Liverpool Transfer Strategy | Goal.Com. [online] Available at: https://www.goal.com/en-gb/news/moneyball-guru-praises-liverpool-transferstrategy/1xys03c1b33tq1gmg94xon8486

[10] - BBC Sport. 2021. Elite Sport Can Continue In England. [online] Available at: <https://www.bbc.co.uk/sport/55534762>

[11] – Timothy, F. (2020). Fantasy Premier League API Endpoints: A Detailed Guide. [online] Medium. Available at: https://medium.com/@frenzelts/fantasy-premier-league-api-endpoints-a-detailed-guide-acbd5598eb19

[12] - Firebase. (2019). Documentation | Firebase. [online] Available at: https://firebase.google.com/docs.

[13] - Premierleague.com. (2018). Premier League Players - Overview & Stats 2018/19 & Archive. [online] Available at: https://www.premierleague.com/players.

[14] - SoccerSTATS.com. (2018). Premier League table, stats, results, form and standings. [online] Available at: https://www.soccerstats.com/latest.asp?league=england

[15] - crontab.guru. (n.d.). Crontab.guru - The cron schedule expression editor. [online] Available at: https://crontab.guru/.

[16] - TheStatsZone.com. (n.d.). The Stats Zone - Sports Data Services. [online] Available at: https://www.thestatszone.com/

[17] - Fantasy Football Hub. (n.d.). Fantasy Premier League Tips by Fantasy Football Hub. [online] Available at: https://fantasyfootballhub.co.uk/

[18] - Football, P. (2020). The stats behind Vardy’s sensational record vs the Premier League big six. [online] Planet Football. Available at: https://www.planetfootball.com/quick-reads/the-stats-behind-vardys-sensational-record-vs-the-premier-league-big-six/

[19] - www.fantasyfootballfix.com. (n.d.). Fantasy Football Fix | Fantasy Premier League tips, price changes and statistics. [online] Available at: https://www.fantasyfootballfix.com/

[20] - DatacoWebsite (n.d.). DatacoWebsite. [online] DatacoWebsite. Available at: https://www.football-dataco.com/

1. https://www.onlinesurveys.ac.uk/ [↑](#footnote-ref-1)
2. https://github.com/ [↑](#footnote-ref-2)
3. https://developer.android.com/studio [↑](#footnote-ref-3)
4. https://firebase.google.com/ [↑](#footnote-ref-4)
5. [↑](#footnote-ref-5)