**A Level Computer Science NEA:**

**1 – Analysis**

**1.1 Background**  
**1.1.1 Problem Statement:**Fantasy Premier League (a.k.a. FPL) is the official “fantasy football” game, which is played by over 11.5 million players across the globe as of 2023. The game is owned and operate by of the Premier League and has been running since the 2002/03 Premier League season. I would like to create a streamlined alternative to the current system, with new quality of life features added. Participants select a team of real-life footballers and earn points based on their performance on the pitch. Every participant (a.k.a. “manager”) has a given budget to spend on players and must make strategic, informed decisions each week about a variety of factors, including transfers, team selection and captaincy. As the aim of FPL is to accumulate as many points as possible over the course of the season, millions of enthusiastic participants worldwide constantly seek new and exciting strategies to optimize their teams. While the FPL app does provide a variety of tools to remain informed on the best players – I believe that there is a gap when it comes to specific, tailor-made recommendations based on the participants own team. Players can spend vast amounts of time simply researching what transfers could be made – and in the end it could all be for nothing when the player doesn’t perform as expected. Due to these facts, I aim to create and FPL companion app that not only has most of the same features that the base app has, but also eases the experience further by being able to recommend transfers, optimal lineups and captaincy choices among other things.   
**1.1.2 Problem Context:**  
For the reader to have the most complete understanding of the problem at hand, there are some pertinent issues regarding FPL that must be discussed.   
For the reader to have the most complete possible understanding of the systems and algorithms that will exist within my solution, and why and how they were written; there are some pertinent issues regarding FPL that need to be discussed.   
**1.1.2.1 Team Selection:**At the start of every Premier League campaign, FPL gives every player in the Premier League a value (with a minimum of 4.0 and a record high of 14.0), this value can change throughout the season based on how many people are transferring that player in/out of their teams. Also, at the start of the season, every participant is given a budget of 100.0, from which they must select a team of 15 players, with the following number of players in each position.  
{Goalkeeper: 2, Defender: 5, Midfielder:5, Forward:3}   
Players may also not have more than 3 players from the same team. Similarly, to player values, player positions are decided by FPL at the start of the season, and do not change over the course of the season – even if a player is being played in a position which does not match their position in FPL.   
Transfers in FPL, are when a participant trades one of the players in their squad, for another player in the FPL database, in order for a player to be able to be transferred in:   
***Total buying cost of players transferred in =< Spare Budget + Total selling cost of players transferred out***  
With rounds of Premier League fixtures organized into “gameweeks”, before the start of the first gameweek, unlimited transfers can be made – however each gameweek has a deadline, where after that point no more transfer can be made – after the deadline for Gameweek 1, a participant no longer has unlimited transfers. A participant acquires 1 free transfer per gameweek, with a maximum of 2 free transfers being stored by a participant at any given time. Any transfers made by a participant will be applied upon the deadline of the next gameweek.   
In terms of the rules regarding a user’s lineup, just like football, a participant may only start 11 players at once, with these being the only 11 players who the participant receives points from. They can be any combination of a participants 15 players if they adhere to the following rules.   
**Goalkeeper:** Minimum of 1, Maximum of 2  
**Defender:** Minimum of 3, Maximum of 5   
**Midfielder:** Minimum of 2, Maximum of 5,   
**Forward:** Minimum of 1, Maximum of 3  
Players can also select a **Captain** and **Vice-captain** for their lineup, any points that a player with captaincy scores, will be counted as double (so if your captain scores 8 points in a given gameweek you will receive 16 points in return). However, if the player that you captained does not play a single minute of football in that gameweek, then the captaincy effect will be applied to the vice-captain of your team, this is the only function of the vice-captain.  **1.1.2.2 Points Scoring:**In the regular points scoring system, players score points based off several actions that they take upon the pitch:

|  |  |
| --- | --- |
| ***Action*** | ***Points*** |
| For playing up to 60 minutes | 1 |
| For playing 60 minutes or more (excluding stoppage time) | 2 |
| For each goal scored by a goalkeeper or defender | 6 |
| For each goal scored by a midfielder | 5 |
| For each goal scored by a forward | 4 |
| For each assist for a goal | 3 |
| For each clean sheet by a goalkeeper or defender | 4 |
| For each clean sheet by a midfielder | 1 |
| For every 3 shots saved by a goalkeeper | 1 |
| For each penalty saved by a goalkeeper | 5 |
| For each penalty missed | -2 |
| Bonus points for the best players in a match (see 1.2.2.2) | 1-3 |
| For every 2 goals conceded by a goalkeeper or defender | -1 |
| For each yellow card | -1 |
| For each red card | -3 |
| For each own goal | -2 |

**Clean Sheets:** If a player is substituted after the 60th minute and they had a clean sheet up until that point, they will keep their clean sheet points no matter how many goals their team concedes.  
**Assists:** Assists are awarded to the player that makes the final pass before a goal is scored (whether intentional or unintentional). If the trajectory of the pass is significantly altered by an opposition player before the goal is scored, then there is no assist awarded. If a shot on goal is blocked by an opposition player, saved by the keeper, or hits the woodwork and a goal is scored from the rebound, an assist is awarded to the first shooter. If a player shoots or passes the ball, forcing an opposition player to score an own goal, then an assist is awarded. The player who is fouled to earn a penalty will be awarded an assist, providing he is not the player who takes the penalty.   
The **Bonus Points System (BPS)** utilizes a range of statistics to create a BPS score for every player in each match. The three best performing players in the match receive the following number of bonus points each:   
**{1st: 3 Points, 2nd: 2 Points, 3rd: 1 Point}**   
**1.1.2.3 Chips:**Chips are essentially ‘power ups’ or ‘buffs’ that participants can apply to their team for any given gameweek across the season. However, they must be applied before the deadline for that gameweek has passed. Each chip may only be used once in a season. There are four in total, Triple Captain, Free Hit, Bench Boost and Wildcard.   
**Triple Captain:** Triple Captain means that a participants captain’s points are tripled instead of doubled in the next gameweek.   
**Free Hit:** Allows a participant to make unlimited transfers for a single gameweek, at the next deadline their squad is returned to how it was at the start of the gameweek.   
**Wildcard:** Means that all transfers (including those that have already been made) in the current gameweek are free of charge.   
**Bench Boost:** The points scored by the players on your bench (the 4 players that were not included in your starting 11 from the squad of 15), are added to your total points for the next gameweek.

**1.1.3 Evaluation of current system:**While I deeply respect the expertise of a team of seasoned developers and software engineers – and in no way can I compete with them in most regards – I believe there is room for innovation. My goal is to include distinctive features in my companion app that have been overlooked by the team at FPL. To ensure my app is as feature-complete and comprehensive as possible, the following is an in-depth examination, and in turn evaluation, of the existing system, enabling me to discern its strengths and areas that may require improvement.   
**1.1.3.1 User Interface:**In general, the FPL website embodies numerous user-interface design principles that could be lent upon when creating my own companion app. Its navigation is quite straightforward, owing to a well-structured and logically organized layout allowing users to seamlessly switch between core features. In addition, the design is clean and maintains a consistent aesthetic, allowing users to focus on the core functionalities – however unfortunately there is no option whatsoever for customization of the design. Another aspect of the design that warrants attention, although it is not necessarily a user-interface design flaw, is the extensive display of information and data on the FPL website. Of which the sheer volume and complexity can be overwhelming for new users. Given that my companion app will also feature a wealth of statistics, it’s important that I remember to keep the presentation of said data user-friendly and easy to read/understand.

**A screenshot of a football game

Description automatically generated**

Transitioning to the team-view screen, the design remains clean and consistent. One notable feature is the clear distinction between the players in the starting XI and those on the bench, a feature which I would look to integrate into my user-interface. The inclusion of both a ‘formation’ view and a ‘list’ view, allowing users who have a specific preference to decide for themselves is also a great addition. Additionally, the interactive buttons enabling substitutions (whilst also providing essential data about players e.g., next fixture, injury status...), add a layer of user engagement that I would be keen to implement in my system. Once again, an area where the user-interface falls short is customization, of which there is no option – something I should look to remedy in my system.

**A screenshot of a computer

Description automatically generatedA screenshot of a football game

Description automatically generated**

Finally, delving into the league-view screen, this screen serves as a hub for the leagues that the user is in (whether public or private), allowing them to view the standings of said leagues. The design remains consistent with the rest of the website: clean, streamlined and without too much clutter. Whilst the current display does offer basic, essential data, it could benefit from more information. For instance, being able to see another manager’s MVP (Most Valuable Player) for that week could be an invaluable insight for the user. Moreover, the screen seems to miss out on some ‘just for fun’ or ‘bragging rights’ features – perhaps there could be a ‘Manager of the Month’ badge that appears next to a manager when they have accumulated the most points within a given month. This feature would recognize outstanding performances, whilst also playing into the competitive landscape of FPL.

**A screenshot of a computer

Description automatically generated**

**1.1.3.2 User Experience:**The user-interface’s intuitive navigation significantly enhances the user experience, and the swift data loading ensures a smooth and seamless journey. Moreover, the real-time updates during matches heighten the immersion, making users feel intimately connected to their teams and intensifying the competitive thrill of the experience – as you can see in real time how players actions are affecting yours and other’s teams. However, the presentation of abundant data without sufficient contextual explanation could potentially confound users (especially users who are newer to FPL), slightly detracting from an otherwise streamlined experience.  **1.1.3.3 Features:**The fundamental features that the FPL website includes are:   
- Users are able view how many points you scored, the average points, and the highest points in the current/most recent gameweek.   
- Users can view their team, including information such as injury status or the upcoming fixture for each individual player.   
- Users can view several simple player statistics for every player in the FPL database.   
- Users can see when the next transfer deadline is  
- Users can view the leagues that they are in, the standings of each of these leagues, and their position in said standings.  **1.1.3.4 Conclusion:**After investigating the FPL website, I can articulate some of the websites strengths:   
  
**1.2 Research**   
**1.2.1 Survey:**The below section includes the preparation for and the results of a survey that was filled out by members of the FPL community. I prepared questions that I thought would guide me in the design of this system. Most of the questions are oriented around the usage patterns of FPL participants to infer what would be required out of a new system – however I also ask direct questions about what they would require from a new system.   
**1.2.1.1 Survey Preparation:**

|  |  |
| --- | --- |
| **Question:** | **Reason:** |
| Do you play in “mini leagues” with friends or colleagues? | If many people play in mini leagues, it could be important to add features where you can view said leagues. Could also introduce features that allow the user to see how their team compares to others in the league. |
| How often do you currently make transfers in FPL? | Should give a good indication of how often player data will need to be updated on our database. |
| Do you use any “FPL-focused” sources of information to inform your transfer decisions? | To see whether players are already trying to make informed decisions in FPL, and if not, the transfer optimisation could be a welcome feature as it does lots of the heavy lifting for the user. |
| Do you find it challenging to decide which players to transfer in/out? | Helps to indicate whether there is a demand for the transfer optimisation feature. |
| What attributes do you find to be most important when judging a player in FPL? | This question should help in deciding how our system will judge a player’s value. The player rating algorithm will be discussed further at a later point, however the information gathered from this question will be considered when deciding how this algorithm is going to function. |
| Would you rather have transfer recommendations that were personalised to your own team or general recommendations for players to look out for? | Should give a better idea of how the transfer optimisation feature should function, as in whether we must view the users team before making recommendations or not – which of course would take up more API calls. |
| How much budget do you usually leave in your FPL balance? | This question will help us find out how much budget should be left over when transferring players in/out of an FPL lineup. However, if these results are inconclusive then a feature could be introduced to allow the user to decide how much budget they want to be left over instead. |

**1.2.1.2 Survey Results:**   
The following is the results of the survey; it was created using Microsoft Forms and distributed by QR code in person to FPL participants in my school.   
A blue and orange pie chart

Description automatically generated  
Based on our sample of FPL players, it’s evident that most FPL players play in leagues with their friends and acquaintances – rather than just playing in the generic leagues provided by FPL. Due to this trend, it would be beneficial to our system to introduce features that enable users to easily access and view the leagues that they are in. A tool to compare your team to others in that league could also be introduced, however I believe this to be less pertinent.   
A colorful pie chart with text

Description automatically generated  
From the survey results, it’s clear that a significant portion of the FPL community actively engages in player transfers. Whilst a minor segment refrains from making any transfers at all, I feel safe in considering them to be outsiders as their numbers are so limited. In addition, incorporating well implemented transfer features in our app may motivate this minority to participate in player transfers.   
A screenshot of a computer

Description automatically generated  
The survey results indicate that while a notable segment of the FPL community does actively seek out FPL-specific information sources, as expected, the majority do not. Thus, indicating a potential market opportunity for a system of transfer optimization, which our system could capitalize on.   
A screenshot of a graph

Description automatically generated  
Supporting the insights from the previous question, the survey data from this question suggests that a significant number of FPL participants tend to struggle with transfer decisions – reinforcing the demand for transfer optimization tools, affirming the decision to incorporate such a feature into the system.   
A graph with blue squares

Description automatically generated with medium confidence  
In this question, respondents were prompted to rank the listed attributes in order of importance to them when they are evaluating FPL players. This feedback will be instrumental in creating the weighted algorithm that will be employed by our system to rate players. Since this system is being made for the FPL community, their input is invaluable.   
A blue and orange pie chart

Description automatically generated  
The survey results to this question indicate a preference within the FPL community for a personalized transfer recommendation system, specifically tailored to individual teams, rather than a generic one that merely suggests high-performing or undervalued players. This emphasizes the importance of implementing a personalized recommendations feature. Nevertheless, in the interest of ensuring a comprehensive and feature-complete system, I will also plan to implement a more generic recommendation mechanism (in order to cater to the minority that favor it)   
A screenshot of a computer

Description automatically generated  
Whilst the survey results from this question suggest that most FPL users typically maintain a balance of 0.0-1.0 in their FPL account, given the considerable variance in responses, it would be more prudent to allow users to set their preferred balance – in the interest of ensuring a more personalized experience.   
**1.2.1.3 Conclusion:**   
This survey was invaluable, providing a deep insight into the FPL community and helping shine a light on their preferences regarding features for an FPL companion app. The following are some of the key points that I picked up on:   
**Leagues:** Functionality for viewing leagues should most certainly be included and the implementation of a system that allows users to compare their teams to others in their league should be considered.   
**Transfers:** Based on the survey results, there’s a clear demand for a transfer recommendation system, given the challenges many FPL users face in making transfer decisions. Given the frequency with which FPL participants make transfers, it’s imperative that our player data remains as current as possible. While I lack a dedicated server, the solution could be to refresh the database each time a user initiates the program. This approach, involving a straightforward API call followed by database write, ensures accuracy without being resource intensive.   
**Player Rating:** The insights derived from this survey, which highlight the criteria FPL users prioritize when assessing players, will be pertinent in the creation of the weighted algorithm that will be used to rate players.   
**Budget Limitations:** Rather than predetermining the remaining budget for users during transfers or team creation, the system will allow users to set their own budget constraints (within reasonable limits). This approach aims to ensure a more tailored and individual experience for each user.   
**1.2.2 Data Analysis:**  
To offer reliable advice to users regarding FPL, the system should utilize a significant amount of data analysis – ensuring that decisions aren’t made based on “gut instinct” or intuition, but rather on solid evidence and observed patterns. Most of the data will be sourced from the FPL API, (https://fantasy.premierleague.com/api/), which provides a variety of endpoints including areas like player and team data.   
In the system, the data will be represented in an SQL database, with objects being initialized in the code when they are required, using the data from the database to do so. The data from the FPL API is represented in the form of JSON, meaning it will need to be parsed for the relevant data and then committed to my own SQL database as previously stated.   
**1.2.2.1 Player Performance Analysis:**   
In terms of player information and statistics, the FPL API offers a wide variety of information about the players in the FPL database, such as:   
**Player Name:** Including options for “First Name”, “Last Name” and “Web Name” (Web name referring to the name used when representing the player in a lineup).   
**Player ID:** Every player is given a unique ID the start of the season.  
**Team:** The real-life team that a player plays for (is a number 1-20, with each number representing a different team in the Premier League).   
**Position:** Is a number (1-4) representing the position that a player is attributed to in FPL.   
**Form:** The form of a player is a numerical value representing their mean points per match in all the games that their team played in the last 30 days (regardless of whether they played in all those matches or not).   
**Total Points:** The total points are the number of points that the has accumulated over the course of the season.   
**Points per Game:** The points per game is calculated as follows ***Player total Points/Player Games Played***  
**Cost:** The cost of a player is the cost of buying a player on the transfer market.   
**Selling Cost:** The selling cost of a player is the value that a user would receive when trading a player on the transfer market.   
To generate a comprehensive rating for each player, these data points will be integrated into a weighted algorithm – resulting in a single numerical value for each player, their **composite score.** Not only does this streamlined scoring system simplify player assessment for the user – but also empowers our platform to suggest informed transfer and lineup decisions.  
**1.2.2.2 Fixture Difficulty Analysis:**While the FPL API provides a Fixture Difficulty Rating (FDR) each gameweek for every team, I would like to at least contemplate the idea of the creation of my own metric specifically designed around the FPL’s points-scoring mechanics. The primary objective being to evaluate the average FPL points a team tends to concede to individual players, rather than the challenge a whole team faces when playing said team. However, this all depends on whether this is possible using the data provided to us by the FPL API – which will be investigated in the Fixture Difficulty Rating Algorithm section.   
**1.2.3 Algorithmic Design:**   
**1.2.3.1 Player Rating Algorithm:**For the Player rating algorithm, I plan to employ a weighted system. The algorithm will combine various player statistics and generate a “composite score” in the form of a floating-point number. It is also important to note, however, that the algorithm may require multiple iterations. This is because while regular transfer and wildcard suggestions should consider projections for several upcoming matches, free-hit recommendations will need to focus on only the next match. These two separate algorithms ensure that our recommendations are suited to the different strategies employed by FPL participants.   
In practice, a dictionary data structure will be used to store the weights for each relevant player statistic – allowing for an efficient lookup and modification of weights. The following is the essence of this method:   
**composite\_score = sum([getarr(player\_information, key)\*player\_weights[key] for key in player\_weights.keys()])**  
The following is a breakdown of the techniques to be used in this solution:  
1. **Dictionary (player\_weights):** This dictionary indicates the importance/weight of each player statistic.  
2. **getattr():** Usage of the built-in function ‘getattr’ is to dynamically retrieve attributes from ‘player\_information’. This is a powerful tool as it allows us to access the object attributes that relate to the keys in the dictionary ‘player\_weights’.   
3. **List Comprehension:** List comprehension is a python feature that allows us to create lists in a concise and clean manner. In this instance, it is used to create a list of products (each player statistic multiplied by its associated weight), which are then added together by the inbuilt function ‘sum’ to create a total composite score.   
For the algorithm to function seamlessly (without error), it’s important that the key names in ‘player\_weights’ are consistent with the attribute names in ‘player\_information’ – any discrepancy could lead to errors that would need to be handled at runtime.  **1.2.3.2 Transfer Recommendation Algorithm:**When it comes to suggesting player transfers tailored to an individual user’s team, several strategies can be pursued. I have identified three primary approaches, some of which would use the player rating algorithm to assist them, to this challenge which I will detail as follows:   
**Iterative Approach:** Utilizing an iterative approach would entail systematically comparing (using their composite score provided by the player rating algorithm) each player in the FPL database against every individual player in the user’s current team. At first glance, this method may seem to be comprehensive, ensuring that no transfers are overlooked, surely this method must uncover some diamonds in the rough? However, upon further inspection, given the large number of players in the FPL database, this approach rapidly becomes increasingly slow to compute. In addition, not all comparisons would be relevant (comparing a Defender to a Midfielder for instance) meaning that extra logic would be required to be put in place further complicating and decreasing the speed of this solution. The delayed response that this approach would create might be seen as unacceptable to user’s pushing them away from the system – so for that reason I will not utilize an iterative approach when creating this algorithm.   
**Greedy Approach:** An approach grounded in the Greedy Algorithm would not utilize the composite score produced by the player rating algorithm. This stems from the inherent nature of greedy algorithms, as they are designed to pinpoint the best possible option in the immediate present. Within the context of the user’s FPL team, the greedy algorithm would target the most glaring vulnerability – the player/s who accumulated the fewest points in the most recent gameweek. With this information, the algorithm would then parse through the pool of available players, choosing the player/s who scored the most points in the most recent gameweek – if this player still allows the user’s team to adhere to selection regulations. It should be noted that to make sure that the player selected adheres to the FPL team selection regulations, additional logic would have to be appended to the algorithm, ensuring that all recommendations are compliant with the game’s rules.   
Although the Greedy Algorithm would be significantly faster than the iterative approach, there are numerous problems that are caused by it:   
1. **Lack of Future-planning:** The greedy algorithm only focuses on the immediate recent past, therefore it may miss out on players that may have been injured in the past gameweek or have a favourable run of fixtures coming up. Additionally, just because a player performed well in the most recent gameweek, doesn’t mean that they will perform well in the next – or any gameweeks after the next for that matter.   
2. **Potential for impulsive reactions:** By basing transfers simply based on the last gameweek, the greedy algorithm could suggest that the user transfer out a consistent high-performer based on just a single gameweek where they did not perform or didn’t play – this could mean that the user misses out on the high-performers high points tallies in the future.   
In conclusion, although the greedy approach is very fast and it can often make good recommendations that are successful, however it is not particularly suited to the long-term planning that a lot of FPL participants like to employ.   
**Linear Programming Approach:** By utilizing the composite score generated for each player in the FPL database by the player rating algorithm, we can frame this problem as a mathematical model. Where the primary objective is to maximize the total composite score while adhering to a set of constraints. These constraints include:   
**Budget Constraints:** The total cost of incoming players should not surpass the combined cost of any outgoing players plus any spare budget that the user has available.  **Position and Formation Constraints:** In accordance with the FPL’s position restrictions detailed in section 1.1.2.1, any player being transferred out must be replaced by a player of the same position.   
**Team Representation Constraints:** A user’s team can have no more than three players from the same real-life football club. All transfer suggestions must adhere to this rule.   
d  
**1.2.3.3 Optimal Lineup Algorithms:**This algorithm will leverage similar linear programming techniques to those used in the Transfer Recommendation algorithm, however it’s implementation will be far more straightforward. Since there is no need to consider the user’s existing team – its primary objective is to determine the best possible squad for the next gameweek. Working within the standard FPL team constraints (budget, player positions and team quotas), the algorithm impartially assesses each player in the database to maximize the composite score.   
Once again, I will not delve into the complexities of this algorithm at this point, as will be done in the design section.   
**1.2.3.4 Fixture Difficulty Rating Algorithm:**  
**1.2.4 User-Interface Design:**When it comes to user-interface design the main, pivotal decision revolves around the choice of the GUI framework that I will use to craft my UI. Three leading contenders in the Python GUI framework sphere are PyQt, Kivy and Tkinter. In the ensuing section, I’ll delve into the strengths and weaknesses of each of these platforms and ascertain the most suitable choice for this project.   
**1.2.4.1 PyQt6:**PyQt6 is an implementation of the Qt framework for C++, used for creating graphical user-interfaces and applications.   
**Advantages:**- Supports all major desktop operating systems, allowing development of applications that can be run on multiple platforms seamlessly.   
- Allows for a high level of customization in the user-interface.  
- There exists a lot of support and documentation online for both the Qt framework and the PyQt6 library.   
- Has a native look and feel, leading to a more responsive and efficient user experience.   
**Disadvantages:**- A commercial license is required for non-open-source applications.  
- Harder to integrate into a mobile app.   
- Does not receive updates as frequently as other UI-frameworks.  **1.2.4.2 Kivy:**Kivy is a python UI-framework that allows for the cross-platform development of GUI apps – although mainly used for developing multitouch applications. This fact makes it well-suited to creating interactive applications across a variety of platforms.   
**Advantages:**- Kivy applications can be run on a variety of different platforms with little to no changes, allowing a larger audience to be reached.  
- Multitouch is supported out-the-box allowing for developing modern, sleek applications.  
- Kivy has an active and supportive community, meaning troubleshooting and getting help online should be a simple task.   
- Kivy is GPU-accelerated, meaning it utilizes the power of the GPU, resulting in smoother and more responsive graphics – providing a more seamless experience.   
**Disadvantages:**- Performance can be slightly worse than in native applications (such as those made by using Tkinter or PyQt6).   
- Kivy is also slightly harder to learn than other python GUI frameworks, meaning it would take more time to develop using Kivy.  **1.2.4.3 Tkinter:**Tkinter is the base GUI framework for python, providing developers with a rudimentary toolkit to create simple graphical user-interfaces. The Tkinter library is included with all python installations.   
**Advantages:**- Straightforward and easy to learn, potentially cutting development time.   
- Does not require any installation of new modules or libraries as it is included with every python installation.  
- Supports all major desktop operating systems.  
- Non-performance intensive, allowing for a responsive and lightweight application.   
- Benefits from a wide community to offer support and extensive documentation.  **Disadvantages:**- Limited number of features and options when compared to other libraries, with a more limited set of a widgets available to the developer.   
- The look of Tkinter is slightly outdated, meaning it is not the best choice for developing modern applications.   
- Less options for customization than other GUI applications, so an application made using Tkinter may look more generic than an application made with another UI framework.   
**1.2.5 Testing Strategy:   
1.2.6 Alternative Solutions:**  
**1.3 Requirements**  
**1.3.1 Resource Requirements:**In order to ensure a smooth user experience, the following is required:   
- A **secure internet connection** is required to ensure consistent access to the FPL API and the system’s database.   
- **Python 3.6 and the included libraries** are also required, however this can be provided in the installation process (e.g., via Github)   
- In terms of **memory**, it is recommended that users have at least 1GB of free RAM, although the actual consumption may be significantly lower.   
- An initial **free disk space** of 1GB is recommended for the installation and operation of the application, as this is the standard amount of disk space required for python and the subsequent libraries I intend to use.   
- **Basic graphics processing** is also a requirement; however, this should not be a problem for most modern computers.   
**1.3.2 User-Oriented Requirements:**The following requirements have been decided upon based on the conclusions I have drawn from the research conducted in 1.2 and my own personal experience and grievances with the base FPL website. In no specific order users of the system should be able to:

|  |  |
| --- | --- |
| Requirements | Reasons |
| Users must be able to log in to their FPL account using their email and password | For personalised lineup and transfer data to be accessed, the request must be sent with the cookies that show that the user has logged in. |
| Users must be able to view their current rank in the ‘Overall’ league in the main window | This is a feature that is available in some alternative solutions, which is more convenient than having to check the overall league’s standings, so this will be implemented. |
| Users must be able to view their current FPL team | This is a feature that is available in the actual FPL app and near all alternative solutions, this is a necessary feature for the application as it is essentially what this application revolves around (lineup and transfer optimisation). |
| Users must be able to view the optimal lineup for the next gameweek | Users must be able to view the optimal lineup for the next week, so that they can inform their own transfer and lineup decisions if they do not want to put complete trust into the transfer and lineup suggestion algorithms. |
| Users must be able to view the optimal lineup for the next 3 gameweeks | Like ­­­­the above, however this incorporates more long-term thinking into the thought process. |
|  |  |
|  |  |
| Users must be able to request to be recommended transfers (and be supplied with recommendations in turn) | This is the main, headline feature of the application, and so it is a requirement that it be present in some capacity; in order to provide the best possible experience for the user. |
| Users must be able to view all the leagues that they are in, how many players each league has, the standings of each league, and their rank in said standings | This is a key feature that exists in the actual FPL app and most alternative solutions, making it a necessity for the application. |
| Users must be able to access the player database and must be able to sort and filter the database according to a variety of metrics (including composite score) | If users do not want to put complete trust into the optimal lineup and transfer recommendation algorithms, then they can view the data used in these algorithms to better inform their own transfer decisions. |
| Users must be able to change the colour scheme, font used, and the sound buttons make (and the volume of the button’s sound) | Customisation of the user-interface is a key feature that is missing the FPL app and most alternative solutions, therefore implementing it into my application is a key feature that differentiates it from the alternatives. |
| Users must be able to log out | Users must be able to log out so that they are able to log in with another account for instance. |
| Users must be able to exit the application without clicking the ‘X’ that is in the top right of the application by default | I want the application to full and whole, requiring it to be able to be closed without the use of native operating system tools. |

**1.4 Planning**   
**1.4.1 Risk Assessment:   
1.4.1.1 Data Security Risks:**Given that the application will likely handle sensitive user information, including email addresses and passwords – which unlock the user’s FPL account – it is vital uphold strong security protocols. This can be achieved by not storing raw, unencrypted data. To fortify the data that is being stored, I plan to use advanced security techniques, such as cryptographic hashing and the addition of salting – to ensure user information is not vulnerable to data breaches. I will go more in depth into the hashing algorithms I plan to utilize in the design section.  **1.4.1.2 API Issues:**Due to my systems reliance on the FPL API (a third-party service), there is an unpredictable nature regarding the API’s uptime and reliability. Should the API face interruptions in service, it is pivotal that the system manages said disruptions with grace, and minimal impact on the user experience. As a contingency, the systems database can be used as a fallback, providing data that is not necessarily as current and up to date as it would be from the API, but remains usable in the absence of data from the API.   
**1.4.2 Time Management:**Effective time management is paramount for success in any long-term project. In this project, I intend to use the Agile development methodology, with a special emphasis on the concept of sprints. Not only aiding me with efficient time allocation, but also ensuring the system produced is of the desired quality and functionality. Agile development is iterative and embraces the idea of incremental improvement, ensuring that at any given point in the development of the software, it remains potentially deployable. Through the integration of sprints – a period within which specific tasks must be completed – I have a structured approach to development. Each sprint offers an opportunity to reflect on the code that was written in the sprint, this is because I plan to undertake bi-weekly sprints. This means that every other week, the work I will be doing is mostly making sure the code is efficiently written and there are no unexpected bugs within the system.   
**1.4.3 Ethical and Legal Considerations:   
1.4.3.1 Ethical:**In terms of ethical considerations, there are numerous possibilities and scenarios that need to be considered:   
**User Consent:** It is crucial to obtain user consent prior to collecting, processing or storing their data. Data collection should also be minimized to essential information only, and stringent security measures should be put in place to prevent unauthorized access.   
**Misinformation:** To prevent misleading users, it is imperative that all FPL-related data displayed is current and accurate. If real time updates are not feasible, it must be explicitly communicated to users that the presented data may not be the latest or entirely accurate.   
**Transparency:** For the sake of clarity and trust, the system should explicitly outline its differences from the official FPL website, including any shortcomings or features it may lack in comparison.  **1.4.3.2 Legal:**In terms of legal considerations there are several to consider:   
**Data Protection Laws:** The system must comply with the Data Protection Act (2018), and as such, clear documentation must be provided to users of the system detailing to them how their data will be used, stored and protected.  
**Intellectual Property:** It must be insured that the content in the system is not infringing on any trademarks, copyrights or intellectual property. Due to this, my system will not use any imagery from Premier League clubs such as logos or kits.   
**Third-Party APIs:** Leveraging the FPL API requires a strict following of its terms of service. Excessive requests to the API could lead to access being revoked, for this reason I will ensure that API calls are kept to a minimum, to ensure consistent and sustainable API access.   
**Age Restrictions:** In alignment with the Premier League terms of service, individuals under the age of 14 are prohibited from registering an account. To ensure compliance with these regulations, I will prompt users to provide their age upon their initial user of the program.

**2 – Design**  
**2.1 Introduction**The contents of this section centers both on the visual design and fundamental mechanics of my system. This will be of use to me as it will supply me with a template from which I can successfully satisfy the requirements of my system – as they were stated in Section 1.3.   
**2.2 Chosen Solutions   
2.2.1 Programming Languages:**My system’s core programs will be made using Python3, leveraging my familiarity with the language to enhance the development process. In addition, Python’s variety of user-interface frameworks make it an ideal choice for creating native user-interface applications. This approach eliminates the necessity to learn new languages, libraries or and frameworks from scratch, streamlining the development cycle, allowing for more efficient progression.   
In addition to Python, I will be utilizing SQL (Structured Query Language) to efficiently access and manage the system’s database. This choice being crucial for the storage, retrieval and analysis of both player and team data – due to SQL’s robust and efficient data handling capabilities. **2.2.2 User-interface framework:**For the user-interface framework of my application, I have elected to use PyQt6, namely for its native yet modern appearance, coupled with its extensive customization options. My decisions stems from a personal preference for PyQt6’s visual style when compared with Tkinter and Kivy, as well as its richer support network, comprehensive documentation, and wide array of tools available online. These attributes will be crucial to a more efficient development cycle, aiming to enhance the development experience and the quality of the finished product.  **2.2.3 Database:**In selecting a database for my project, I’ve opted for Microsoft’s SQL Server, accessed and managed via SQL Server Management Studio (SSMS). This choice being primarily driven by its local deployment capability, providing a more cost-effective solution when compared with cloud-based alternatives. Additionally, the inherent flexibility of SQL allows for the easy migration of data to a cloud-based solution in the future, should a public rollout of the application deem this necessary. These two main advantages of cost-efficiency and scalability make SQL Server an ideal choice for the data storage needs of the application.

**2.3 System Architecture  
2.3.1 UI Architecture:**The initial interface presented to users upon accessing the application will be the login page. From here, users can authenticate using their FPL account information, with the implemented login features. Upon successful logon users will be redirected to the main page of the application.   
The main page servers as the centralized hub of the application, providing an intuitive user interface for users to transfer seamlessly between the apps core functionalities and features. In addition, an Exit Page is accessible from both the Main Page and Login Page, providing a straightforward option for users to exit the application.   
The application is designed with three core windows, each dedicated to one of the system’s main functionalities. The first, the Lineup Page, enables users to not only view their current lineups, but also request lineup and transfer recommendations to optimize their FPL experience. The second, the Leagues Page, offers users the ability to monitor the leagues they participate in and their standings. Finally, the Player Database page serves as a comprehensive, but also abstracted, player statistics resource allowing users to be better informed when making their own transfer or lineup decisions. This structured approach ensures users possess ease of access to each of the application’s core functionalities – enhancing the user experience.   
The below is a user-interface flow diagram to properly depict the system: A diagram of a company

Description automatically generated **2.3.2 Database Architecture:**My database will consist of two tables, the players table and the teams table:   
**Players Table:** This table is designed to catalog comprehensive data on individual players. It serves as a vital source of data for the algorithms within our system, ensuring efficient and effective operation. The table is structured to facilitate quick storage and retrieval of player information – providing the backbone for player analysis and lineup creation capabilities.   
**Teams Table:** This table is strategically designed to complement the players table, providing information on each real-life football team in the Premier League, of course encompassing all the teams that players play for – and by extension all the teams that players will play against. A pivotal aspect of the teams table is the way it interacts with the players table. The primary key of the teams table ‘Team ID’ is used as a foreign key in the players table, allowing for seamless retrieval of the Fixture-Difficulty-Rating (FDR) for each player, contingent on their respective team. By incorporating this ‘FDR’ into the Player Rating algorithm, the algorithm gains enhanced capability to deliver contextually aware evaluations of player performance, aswell as ‘predictions’ for future performance.   
A screenshot of a computer program

Description automatically generated

**2.4 Data Structures**In my solution I will utilize a variety of data structures, including the following:   
**Array:**   
In my system, arrays will serve as the primary data structure for storing player objects (which are initialized from the player database). By having a structured collection of player objects the Player Rating algorithm, Transfer Recommendation algorithm, and Optimal Lineup algorithm are provided with the dataset that they require to function.   
**Dictionary:**   
In my solution, I will leverage dictionaries to simulate the functionality of ‘SWITCH’ statements, which are not provided natively by Python. By mapping distinct values to respective method calls within a dictionary, I can streamline the execution of different pieces of code based on user input. This method provides me with a clearer and more efficient approach – compared to using nested selection, which could potentially improve the performance of my solution.   
In addition to emulating ‘SWITCH’ statements, dictionaries will also play a crucial role in the algorithms used for calculating composite player ratings. Each attribute determining this rating will be assigned a ‘weight’ in a dictionary, this weight representing the attributes influence on the overall rating. The sum of the multiplications of each attribute’s value and the corresponding weights will provide me with a rating for a given player. Using a dictionary allows for the weights to be dynamically updated, which could be important when tailoring the user experience. Below is a mock-up of what this A computer screen with numbers and symbols

Description automatically generatedsolution could look like:   
I will also employ dictionaries to manage the URLs used when interfacing with the FPL API. Each API endpoint being represented by a unique keyword in the dictionary. This approach allows me to quickly and easily access the URLs without needing to recall or refer to a longer web address – as I will be using multiple endpoints throughout the system. It also enhances readability throughout my code as it is more concise and cleaner, the specific keywords also could provide the reader with a more intuitive understanding of what each part of the code is doing. Finally, having a more centralized storage of API endpoints, the process of updating said endpoints is much faster and simpler – as only the dictionary needs to be updated and not every reference to the endpoint throughout the code.   
**Stack:**  
  
**JSON:**Although not strictly a data structure, I feel it is important to mention how JSON will be used in this system. In the system, JSON plays a pivotal role in the storage of user-interface settings – including a range of personalization options such as color themes, font styles and button sound preferences. Shown A screen shot of a computer code

Description automatically generatedbelow is a sample representation of how said JSON might be structured:  **2.5 Algorithms   
2.5.1 Optimization Algorithms:**[EXPLANATION]   
**2.5.1.1 Transfer Optimization Algorithm:   
2.5.1.2 Optimal Lineup Algorithm:   
2.5.2 Rating Algorithms:   
2.5.3 Sorting Algorithms:**In the `Database Window` of the application, users will be afforded the ability to sort the players in the database based on various key metrics. To integrate this feature efficiently, a quick and robust sorting algorithm is of paramount importance. Due to these requirements, I will implement a merge sort, as it is known for its predictable and reliable performance in addition to its ease of implementation. This choice enhances the user-experience by providing quick and accurate data organization.  **2.5.4 Database Filtering:**To enable users to effectively filter the database by various key metrics, several methods can be implemented:   
**Linear Search with multiple criteria:** Implementing a Linear Search with multiple criteria, though not the most efficient, could adequately serve the system’s needs. This method entails incrementally scanning through the player dataset and identifying the players that meet the specified criteria – with these results being compiled into an array. This array is what will then be displayed in the `Database Window`. This straightforward approach should ensure that all player data is presented accurately and comprehensively.   
**SQL Querying:** To implement filtering through SQL queries, the process typically involves a straightforward `SELECT` statement, structured similarly to the following:   
`SELECT \* FROM tbl\_players WHERE criteria1 = value1 AND criteria2 > value2;`   
This method capitalizes on SQL’s powerful data retrieval capabilities, resulting in it being highly efficient for individual filter operations and ensuring that only relevant player data is returned. However, it is pertinent to consider the implications of repeated use of this feature, with the repeating execution of queries which could potentially impact performance. This is an eventuality that will be explored further during the development process.  **Data-management Libraries:** The Python programming language offers a suite of data manipulation libraries, notably `pandas`, which could potentially significantly streamline the integration of database features into the `Database Window`. Utilizing pandas would first involve transferring the data from the database into a `Pandas Dataframe` - a two-dimensional, tabular data structure which is optimized for data manipulation. This transfer can be easily facilitated through the user of SQL database-specific connectors provided by external libraries.   
Once the data is formatted into a Pandas Dataframe, pandas’ powerful built-in functions can be employed to filter it, ensuring that only pertinent player data is presented to the user. Pandas stands out for its exceptional ease of use and in-memory processing, which would obviously not be available using SQL queries. However, i0t is crucial to acknowledge that to employ pandas would mean that extra steps would have to be undertaken to appropriately format the data to ensure seamless compatibility with the library – this being the main drawback that stems from its usage.  **2.6 User Interface Design   
2.6.1 General Principles:   
2.6.2 Login Window:   
2.6.3 Main Window:   
2.6.4 Exit Window:  
2.6.5 Lineup Window:   
2.6.6 Leagues Window:   
2.6.7 Database Window:   
2.6.8 Settings Window:**   
**2.7 Input and Output**   
**2.8 Error Handling**   
**2.9 Security  
Fix this at some point**Since as of currently, my system only uses user data temporarily (when retrieving the session cookies for a log on), there is not much risk of the stealing of user data. However, in the future, it may be required for a ‘remember me’ feature to be implemented – meaning user passwords must be stored in a database. Whilst such a feature isn’t explicitly planned at this current moment in time, its potential inclusion would greatly simplify user access by eliminating the necessity for repeated logins. However, the implementation of this feature would necessitate securely storing passwords in the system’s database. To achieve this in accordance with the previously mentioned ethical and legal considerations (in 1.4.3), rigorous security measures would have to be utilized. These measures could potentially include a variety of encryption techniques, with the main principle being that user credentials are not stored in plain text. The main encryption techniques that would be used are hashing and salting. The hashing algorithm that I would use in this scenario is Argon2, as it is the only one that I have used in the past (therefore the only one that I am remotely familiar with).   
**2.10 Performance Considerations**In terms of performance, the application should be able to run on most modern low-end systems. Its core is developed in Python, features a non-graphically intensive user-interface, ensuring minimal strain on both the CPU and GPU. However, one potential performance bottleneck is related to memory management, and more specifically the retention of window objects in memory even after they have been closed. This could potentially lead to increased, unnecessary, memory consumption – therefore impacting the applications’ overall performance negatively.   
However, since the PyQt6 library is being used for the UI of the application, the simple and robust solutions for memory management that the library offers can be used to reduce the memory footprint of the application. In fact, this can all be implemented in a single line of code into the parent class of all the windows in the application. As demonstrate by the following:   
  
This line ensures that whenever a window is closed in the application, it is deleted from memory rather than stored until the entire application is closed. This strategy is critical for maintaining the application’s responsiveness and ensuring it can be run-on low-end systems with lower memory capacity.   
**2.11 Testing Strategy**  
**2.12 User Documentation**  
**2.13 Code Structure and Organization**The application will be structured into distinct, modular components, each of which encapsulating a unique set of functionalities. This design allows common code segments that are shared across the codebase to be updated centrally, reducing the effect of unintended consequences on other parts of the codebase. The architecture of the application will be organized into four separate files as follows:   
**utils.py:**The `utils.py` file is designed to be the central storage of for various utility elements that are essential across the program. This includes constants, such as API endpoints and commonly used functions, like fetch functions and sorting algorithms. As mentioned previously, centralizing shared resources simplifies codebase maintenance, as updates or bug fixes are immediately propagated throughout the rest of the system. The existence of this file is also in compliance with the DRY (Don’t repeat yourself) coding principle as code that is used multiple times throughout the application is written once, and utilized wherever else it was needed in the application. Finally, having a dedicated utilities file allows for easy navigation throughout the codebase, especially since modern text editors allow developers to trace used functions back to their original definition.  **base.py:**The `base.py` file is designed to function as the foundational module of the system, storing key data structures, entity definitions and functionalities critical to the application’s process. The file will encapsulate core elements like data classes for entities such as ‘User’, ‘Player’ and ‘League’, ensuring that the representation of these components is standardized throughout the application – ensuring efficient data management and

manipulation. Additionally, `base.py` will include sophisticated algorithms that can be used to optimize player lineups and transfers, based on several key metrics.   
By centralizing these components, and more that have not been mentioned, the development and maintenance of the application becomes much more streamlined and straightforward. Once again, like `utils.py`, this approach is in accordance with the DRY (Don’t repeat yourself) principle, ensuring that the foundational blueprints for entities in the program are stored in a single location – allowing for the dynamic creation of such entities. **fpl.py:**The `fpl.py` file serves as the main module for the operation of most of the features of the application, it is structured to harness the foundational structures and key functionalities defined in `base.py` and `utils.py`. Within the file, the ‘FPL’ class is stored, functioning as the nerve center for all FPL-related functions of the application.   
This class is engineered to facilitate seamless interaction with the FPL API and the application’s own database, leveraging methods and constants from `utils.py` for efficient communication and data exchange with both external and internal services. The key functionalities stored in this class include the retrieval and processing of player information, user lineups and league standings, among others. By centralizing FPL-related operations in this class, the module enhances the maintainability of the codebase – facilitating ease of updates and enhanced readability.  **ui.py:**Finally, the `ui.py` file is crafted to serve as the graphical user-interface (GUI) layer of the application. It utilizes the PyQt6 framework, which is responsible for creating an interactive front-end that facilitates easy navigation to all the applications features.   
The main feature of `ui.py` will be the `WindowParent` class, which will integrate a collection of shared properties and methods that each individual window in the application will require – offering a structured and organized layout throughout the program. Every other window in the application will be inherit from this parent class and will be dynamically generated in the application based on user inputs.   
In essence, the `ui.py` file is the component of the application that brings the visual and interactive elements of the system to life for the user, resulting in an efficient and user-friendly interface.   
**2.14 External Libraries and APIs**My usage of external libraries and APIs will be limited, with the following being used:  **PuLP:** PuLP is a linear programming (LP) modeler for Python, that facilitates the usage of linear optimization within the application. It will be used in both the `TransferOptimiser` and `LineupOptimiser`, to aid the process of suggesting transfers and optimal lineups to the user.  **Mysql.connector:** Mysql.connector is a library used to connect python applications to a MySQL database. It will be utilized in my application to connect to the database that stores player and team information (and potentially user credentials also).  **PyQt6:** PyQt6 is an implementation of the Qt framework (C++), used for creating graphical user-interfaces and applications. It will allow me to create an interactive and sophisticated user-interface for the system.  **FPL API:** The Fantasy Premier League (FPL) API is an unofficial API providing access to the data that the official FPL website and app uses to update the data that is presents. It will be used in my application to provide me with the data to populate my database with player and team data.   
**2.15 Data Flow Diagrams**  
  
**3 – Technical Solution**  
  
**4 – Testing**  
  
**5 – Evaluation**