FPL POINTS PREDICTOR

[Prannvat Singh]
[Maiden Erlegh School]

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Table of Contents

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Analysis

Prannvat Singh

What is the problem?

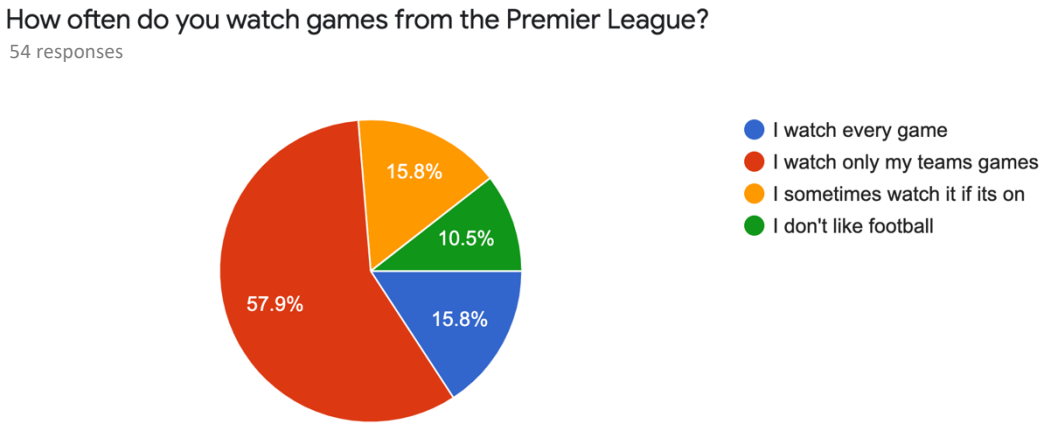
Football is the biggest sport in the world, especially in Europe everyone knows at least one person that is mad about football. The Premier League is the biggest football league in the world and England is the home of the Premier League. Therefore, there are millions of people who watch the Premier League. Thus, in order to bring even more excitement for the fans of the Premier League, in 2002, a very exciting game was introduced, called Fantasy Premier League (FPL), in which everyone got to make their own teams and got rewarded on how the players in their team did week in week out. This game has become so loved that in the 2019-'20 season, over 7.6 million people played this game over the season. Everyone wants to get the most points every season and fight for rewards or just for the satisfaction of beating their mates; so, what if there was a program which could predict, to a high accuracy, the number of points a certain player will get in a certain game week? Is this something people would even be interested in? What would be the demand of this algorithm? Well, these are all questions that needed to be

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

researched. Therefore, through some extensive researching, surveying, and interviewing I gathered numbers to get an idea as to how many people would be interested and what their needs would be.

Research from wide audience

- Firstly, it was crucial to understand if the love for the Premier league was truly as common as it pre-emptively seemed.



It was clear after the results that my assumptions were indeed accurate. 70.5% of people watch football every week, but most only watch their own team's games. The fact that most people don't watch every game would mean that my project would only be more useful, as it is not possible for anyone to know how a player that is not in the team they support is playing; thus, making it difficult for them to pick them in their FPL. With the help of my points predictor, this would no longer be a problem.

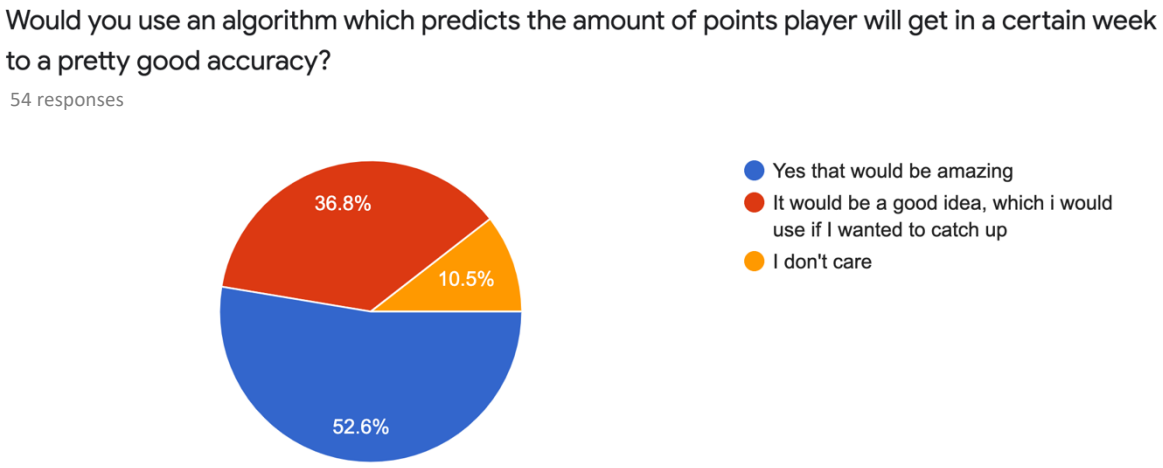
Primarily, my client is my fellow student, Thomas Clark, who has requested me to make this app for him as he feels that he struggles to do well in the FPL leagues he is in with his friends. The interview for this can be seen in {Appendix}. All the objectives and tasks that I talk about in this analysis are based on his request and I am going to make an app which fulfils the needs of my client.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

However, I want to also see if this app can be taken further in the future and if there will be a demand for it.

- Who are my end-users going to be to for my project?

  To gather some end-users, I interviewed my friends and peers and managed to gather around 10-15 people that would be interested in using my product. I also surveyed people and managed to get these results about what they thought about my product and what they would use it for:



  Over 88% of people showed a great amount of interest in my product and said that thy would use it because they thought it was amazing or because it would be useful if they needed to catch up to their opponents in FPL.

Research done about product(also in Appendix)

- Were there any pre-existing products I could take inspiration from?

  I then started researching about products that already existed that were like my idea. It turned out, there were products very similar to my idea already out there, however as I did my research, I also found that there was a serious of appeal in these products. A product like this, in my humble opinion, must have two very important features. One of them being the fact that it predicted points, which of course was there. However, the second thing was that it must have a user interface, which looked pleasant and professional to look at. I found websites

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

that did what my project intends on doing however, as I went through them, it seemed evident to me that they didn't look too appealing to use. For example, this website: {image below}, is functional, but it looks a bit messy. This is something I have learnt from, and I now understand that in order to have a successful product it must be functional and aesthetic. This would all be a part of my UI planning and layout.

✔	Player	for Info	Team	Pos	Cost	Merit	Form	Prob. of Appear-ing	PP GW 26	Val GW 26	PP GW 27	PP Next 2	Val Next 2	PP GW 28	PP Next 3	Val Next 3	PP GW 29	PP Next 4	Val Next 4	PP GW 30	PP Next 5	Val Next 5
	Gabriel		ARS	DEF	5.2	4.11	1.2	0.97	6	2.73	3.8	9.8	4.45	4.9	14.7	6.7	5.4	20.1	9.13	2	22.1	10.04
	Salah		LIV	MID	12.7	6.12	1.04	0.97	5.6	0.61	5.5	11.2	1.21	6.1	17.3	1.88	8.8	26.1	2.83	5.1	31.1	3.39
	Haaland		MCI	FWD	12.2	6.93	1	0.97	5.6	0.64	5.2	10.8	1.24	6.2	17	1.95	6.6	23.6	2.72	5.5	29.1	3.34
	White		ARS	DEF	4.7	3.75	1.07	0.96	5.5	3.25	3.5	9	5.31	4.6	13.6	7.99	4.9	18.5	10.9	1.9	20.4	12

Modelling

In addition to this, I have researched further (links can be seen in {Appendix}, and my plan is to make an app, which has a main menu. This main menu should be clear and have a few options for the user to access different parts of the app. These will include the option to make searches for players and then get a prediction, and also see the current standings of total points by all players I intend to make my app have a nice colour scheme; the colours I intend on using are green and purple, as these are the colours of the Premier League app. This colour theme will bring the user a sense of relatability between the Fantasy Premier League and my app. This will be much better than the {image above} and other sites {Appendix} as it will look appealing and not overwhelm user with random colours and hard to navigate information. I will also incorporate the Premier League logo into my GUI. Upon research I found websites[1] that display the standings of current points and what the predicted points are. However, a lot of these aren't based on a machine learning model, but the predictions of the people that run the website. This will not be the case for my app as my app will make predictions based off of the neural network I will make, upon my client's wishes.

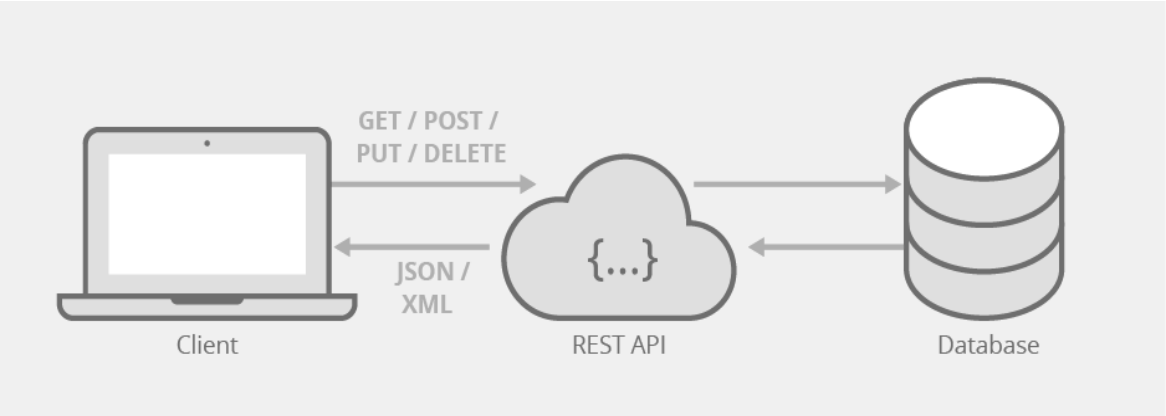
Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

SMART OBJECTIVES

As you can see in the {Appendix}, I had an initial interview with my client, Thomas Clark. However, I also have regular dialogue with him about this product. I asked him about the objectives which enabled me to specifically plan my project according to what he wants in the app.

OBJECTIVE 1

REQUESTING AND ACCESSING THE DATA FROM PREMIER LEAGUE API AND PARSING AS JSON



The data which I would use would have to come from somewhere. After some research I found out that I would have to use an Application Programming Interface (API). This is a software intermediary, which would allow me to 'talk' with the Premier League database. As I talked with my client, I found that he felt that the premier league API was clearly the most reliable way of fetching the data as it would be the latest data, and all verified officially.

Once I get the data, I will have to store the desired data into a table, which is compatible with usage of SQL Once this is done, I will be able to set fields, have unique identifiers for players and then access information about the players with SQL.

To be more specific, I will use JSON parsing to convert the JSON format into text which I can then use as it will be in the form of a dictionary, from which I will be able to make my data frames. There are several endpoints[2] to the main Premier League AP1 and I will be able to use them after some research on what they all store to collect the necessary data.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

This is measurable, as I can easily request and get data from the Premier League API, which is public and free to use, it is also instantly updated with changes to the data. This allows for a great way to gather fresh data at all times and once I begin making the data frames, I will be aware of all the additional information I may need to get very easily.

Therefore, this is very achievable as the API request system is rather simple, even though it is a high-level technique, and with my skillset, I will be able to achieve this efficiently.

Doing this is very relevant to my project as I cannot do the rest of my project like the neural network and GUI without gathering data first.

I aim to have this complete rather quickly relative to my whole project time as I will need the data from the start to then use and add onto my SQL tables and use for my machine learning and GUI.

OBJECTIVE 2

USING SQL TO STORE DATA COLLECTED FROM API, JOIN TABLES AND ACCESS PLAYER INFORMATION FROM

Once I get my data, I can store them as data frames with the help of python's efficient Pandas module. This will allow me to then transfer them easily onto SQL tables. To use SQL, there are several options:

MySQL Workbench – This is a very common and widely used software to handle SQL tables on. This requires a server to run on, which would have to be my device. This may cause problems, because if in the future I ever wish to make this project public on other devices, my device will have to stay on.

SQLite: This is a very good option in my opinion, as it is a server-less database and is self-contained. This means it is an embedded system therefore the databases will run as part of the app. Upon some conversations with my client, a portable database was important to him, and therefore, SQLite is clearly the best option to store my data on.

SQL is a crucial part of my project, using cross-table parameterised SQL and aggregate SQL function will allow me to show average points of players etc. as well as to join

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

tables together and display information about players. Specifically. I will end up having a 3rd Normal Form table, which will be my user details, I will have a table which I will have created by joining two tables together and this will have information such as the player position, how many people selected the player, how many points they got in the previous week etc.

This can be done measurably as once I have my data ready, I can see if I can easily update these tables with new information when needs be, if I have the required information I need and if I can use the smaller tables with higher form of normalisation separately or not.

This is a very realistic aim as it requires all the highest-level skills that I am taught in my A – levels and therefore I can manage to use those skills in this part of my NEA.

Also, for my project, SQL is very relevant as it will allow me to handle a lot of data easily on different tables.

I will need this to be done before the completion of my GUI and machine learning model as I will need to use these tables to extract new data from which I would use my model on, and I will also have a page will output some player information in my GUI. I will plan my database tables in my {Documented Design}.

OBJECTIVE 3

ACCOUNT REGISTRATION- MAKE AN ENCRYPTED AND AUTHENTICATED LOGIN AND REGISTRATION SYSTEM

I will also make an account registration and login system in my app. As I discussed this with my client, we concluded that this would have to be done using defensive programming in order to prevent SQL injections. This will be achieved using parameterised SQL. This objective will allow me to use SQL to store usernames and passwords on an SQL table. This will mean that users will be able to enter the app after registering or logging in. This opens room for further advancements in my app if I wish, which may allow users to store their personal teams in their account. This will require lots of validation, interaction with SQL tables and I will have to ensure that there is no way of accessing the app without the correct details.

Specifically, I will need to make a system which works through my GUI, and allows for users to register into the

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

app, I will make it so no two people can have the same username and that the password be at least 6 characters long. In addition to this, I will also have to encrypt my passwords so that in the case of a data base break-in, my users' sensitive information remains safe. To do this, I will most likely use 'sha256'[3] encryption.

To measure this, I want to have at least two people test my app and register with their own username and password. { Testing Video and Evaluation}. Doing this successfully will mean that my app has the potential to be popular and it will verify that my account system works.

This will be achievable, considering that the basis of an account registration system is quite simple as it just validates inputs, however, with some research I will also do the encryption and store the info into an SQL table.

The relevancy of this to my project is that enables a nice feature for my client which allows them to use their own account and it will also mean that I can make further advancements onto the app if my client ever wishes like I mentioned earlier.

My aim is to have this done as a first step when I begin implementing my GUI and that will be the final step of my whole project.

OBJECTIVE 4

GRAPHICAL USER INTERFACE – MAKING THE BACKEND OF NEURAL NETWORK BE DISPLAYED ON AN APPEALING USER INTERFACE WHICH IS EASY TO USE FOR USER

The way to make a Graphical User Interface (GUI) on Python is to use something like the PySimpleGUI package or the Kivy package. I have many options of what package I can use.

PySimpleGUI[4]: This package, like any other Graphical User Interface (GUI) toolkit, uses something called widgets. A widget is a term used to describe elements that make up the user Interface. For example, for this project that may be something like an image of a player. For an idea of what this would look like, I found a very simple example on the internet, which creates a window and displays a message. Of course, this is very basic and not what my GUI would look like, but it shows the usefulness of PySimpleGUI.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Kivy[5]: This is open-source, and it lets you focus on building completely custom and interactive applications. With Kivy, you can build everything from fully scratch, even if you want a scroll bar for your screen, you can make that yourself. Kivy also allows for complete Object-Oriented-Programming, and this is a great thing for my project.

Tkinter[6]: This package is the standard package used on python. It applies a lot of its own logic itself and isn't completely bare boned for you to build all yourself. I feel like this isn't ideal for my NEA as it won't allow me to display my skills to the highest standard. Also, something like Kivy will allow me to fully customize everything myself.

Overall, after the research on all of these packages from their documentation, I believe that using Kivy will be most beneficial for me as it will allow me to build my GUI from nothing, all by myself and it will also mean all my GUI code will be Object-Oriented, which is crucial for my NEA.

Specifically, I know that I need several different GUI's which can be accessed from one other, and I will have to design these to make it user-friendly and appealing {Documented Design}. I will constantly check with my client-Thomas Clark and other people that show interest in my app about my GUI and my design to see if they like it and what improvements I can make to it. Thomas Clark wishes for the colour scheme to be green and purple as that replicates the real FPL colour scheme.

In addition to this, as I mentioned above, I will need to make this objective measurable and I can easily do this by asking my client what they think of the design {Documented Design} before I start implementing and then I will ask them once I have implemented {Evaluation}, so I can make improvements if needs be.

This will be achievable if I use the Kivy documentation to help me, as this is not something we are taught in A-levels, I will need to research and learn the way Kivy works. For this I will have to do my own learning in my own time, which I can do in order to make a good app for my client.

A good Graphical User Interface is very relevant and crucial for my project as it will give my client the ease he requires, and it will let him check for predictions easily.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

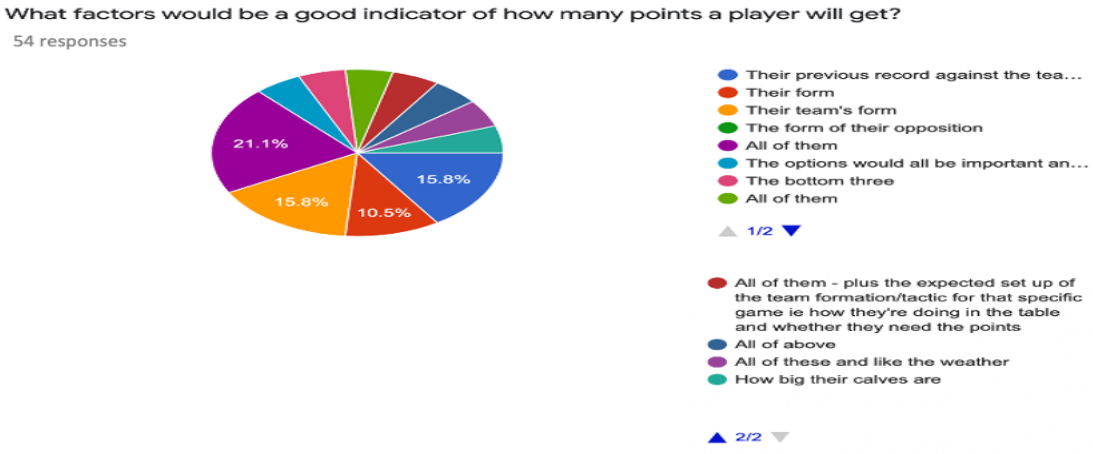
This is an important request for my client because unlike the Objectives 1 and 2, which my client won't have to deal with himself, a GUI is something that my client will use the app through.

I intend to implement my whole GUI after I complete my neural network and my collection of data, as I can then easily implement the backend features to the front-end GUI.

OBJECTIVE 5

MACHINE LEARNING MODEL – COLLECT DATA AND TRAIN A NEURAL NETWORK TO MAKE PREDICTIONS BASED ON NEW DATA ABOUT THE PREDICTED POINTS OF A FOOTBALLER

There are options to be considered when it comes to the use of machine learning. There is a possibility that I could program a Linear Regression Machine Learning model in which the model will be able to determine a line of best fit against teams, using multiple variables such as form, difficulty of opposition etc. I asked my client what he thinks I should train my model based ; in his opinion the best data to train the model on are the amount of people that buy the player, select the player, who the opponent team is, whether they are playing at home or away. This was a very insightful discussion with Thomas and has given me a very god idea of how to implement my model. To get a better idea of what these variables could also be, I asked several people, and this is the response:



Another option would be to using machine learning libraries for Python. Examples of these libraries are SciPy and TensorFlow, Keras, which contains modules for optimization, linear algebra, integration, and statistics. This would allow me to prepare my

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

data, then using the algorithms in the library, examine the results and see what the best way to get results is. Another library I could use is Scikit-learn, which can be used for data mining and data-analysis.

SciPy:

This is an open-source Python library, which is used for scientific computing as well as technical computing. It is very high-level, making it easy to use and it provides algorithms for optimization integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics, and many other classes of problems. It is written in low-level languages like C and Fortran, which mean it is very efficient and fast.

TensorFlow and Keras:

TensorFlow and Keras were created primarily for creating and training neural network models. For building several kinds of neural networks, such as convolutional neural networks, recurrent neural networks, and more, they offer a straightforward and user-friendly interface.

TensorFlow can handle massive datasets and intricate neural network models since it is very scalable. This makes it perfect for developing models at the production level and for training models on huge data.

High-level APIs: Keras is a high-level API built on top of TensorFlow, making neural network model construction easier for users. By removing low-level implementation details, it enables users to concentrate on the model's architecture.

TensorFlow may be used for distributed computing, which means that the task can be divided among several processors or machines. This can shorten the time needed for large models and datasets to train.

TensorFlow is also open-source and is great for making neural networks and machine learning algorithms. Therefore, I believe that Keras on top of TensorFlow is my best option as it has a good interface and due to its high-level-language and specialty in neural networks, it is best suited in my project.

Specifically, once I have collected my data and am ready to use it for training, I will have to trial and error { Documented

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

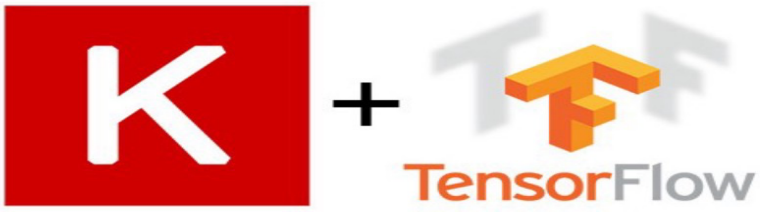
Design, Testing, Evaluation} to see how many layers my neural network graph will need. I will use the rectified linear activation function (ReLU); after some research I found this activation function to be very efficient and useful. I will also use the linear activation function if I end up having more than two layers in my neural network {Documented Design}. I will also have to decide the epoch and batch-size values through trial and error depending on what values give me the best loss value {Documented Design}.

To measure the success of my neural network I would use the loss value, which shows the summation of errors in the model. The lower I can get this to be, the better I am, however if I get it too low, that means my model will be inaccurate cos it will have 'overfit' into the training data thus it will just predict relative to the values in that dataset. I wish to get my loss value to a value below 5 but below 3.

I believe that this is achievable however as machine learning is not in the A-level syllabus, I will have to research and figure out how to make a neural network[7] and make the best model I can using Keras and TensorFlow.

This is very relevant to my project as this is the primary request of my client, and without this my product will not do the main thing that it should, which is predicting the points footballers will get in the FPL.

I hope to get this done right after I have collected the data and with enough time so I can implement my GUI and incorporate everything into the GUI.



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Documented Design

Prannvat Singh

Overview

The design for my project will have many steps in order to create a useable product. Initially, it is important that there is a menu, from which the user can get access to everything else. Also, as well as the menu. As well as this, a search bar is crucial as this will allow users to conveniently find players and information about them.

Also, the user will be able to see the current player forms with all the information about their total points. This will allow for a good UI to be possible as well as a convenient main home page for the product. With the help of machine learning, the data will be used and the predicted points etc. will be shown to the user.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Program Structure

CLASS PROGRAM
CLASS PremierLeagueApp
CLASS GUI
Make all screens classes
Call machine learning model script
Fetch data from SQL database
RUN PROGRAM

The program will have three different scripts in order to keep a tidy environment and have easy to navigate code. The PremierLeagueApp will be where everything will eventually come together and then the user will be able to use the GUI and make searches, see point standings of all players in the premier league and. The PremierLeagueAp will be instantiated and then after the login is verified, the other screen classes will be instantiated in order of when they are accessed.

Login verification Pseudocode

```
START

    FUNCTION loginUser(username, password)

   BEGIN

       IF username AND password are not empty THEN

            IF username EXISTS IN usersTable AND password MATCHES WITH
usersTable[username].password THEN

                PRINT "Login successful. Welcome, " + username

                CALL PremierLeagueApp()

            ELSE

                PRINT "Error: Invalid username or password."

            END IF

        ELSE

            PRINT "Error: Username and password are required."

        END IF

    END FUNCTION
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Mergesort Pseudocode

```
START

    FUNCTION mergeSort(arr)

    BEGIN

        IF LENGTH(arr) <= 1 THEN

            RETURN arr  // Already sorted

        ELSE

            middle = LENGTH(arr) / 2

            left = CALL mergeSort(SUBARRAY(arr, 0, middle))  // Recursive
call for left half

            right = CALL mergeSort(SUBARRAY(arr, middle, LENGTH(arr)))  //
Recursive call for right half

            RETURN CALL merge(left, right)  // Merge the sorted left and
right halves

        END IF

    END FUNCTION

    FUNCTION merge(left, right)

    BEGIN

        result = EMPTY_ARRAY  // Initialize an empty array to store the
merged result

        WHILE left AND right are not empty

        BEGIN

            IF left[0] <= right[0] THEN

                APPEND left[0] TO result  // Take the smaller element from
left

                REMOVE left[0] FROM left

            ELSE
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                APPEND right[0] TO result  // Take the smaller element from
right

                REMOVE right[0] FROM right

            END IF

        END WHILE

        // If there are any remaining elements in left and right, append
them to result

        APPEND left TO result

        APPEND right TO result

        RETURN result

    END FUNCTION

END
```

In order to for my program structure to work, I need to request data from the Premier League API.

API Request Pseudocode

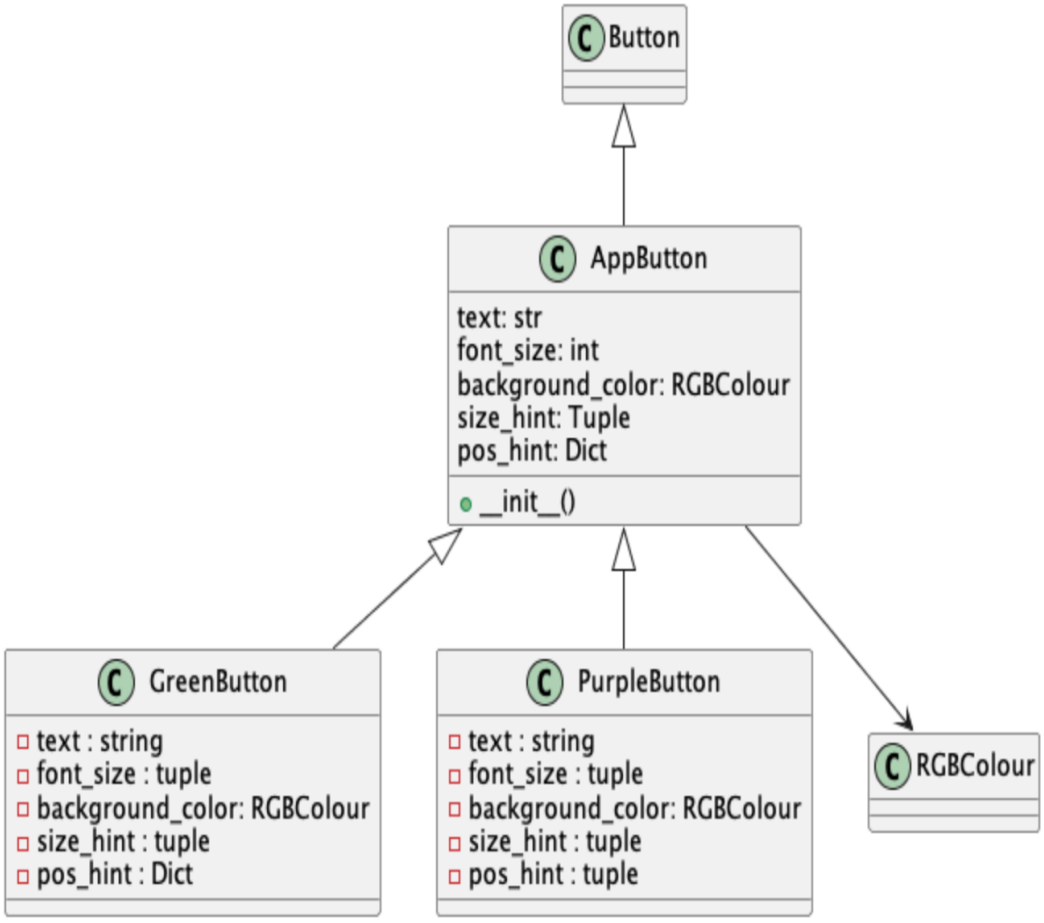
```
START

    FUNCTION requestDataFromPremierLeagueAPI()

    BEGIN

        url = " https://fantasy.premierleague.com/api

       response =  request.get(url)

        IF response.status_code == 200 THEN  // Successful response

            data = response.data  // Extract data from respons

            PRINT "Data from Premier League API stored successfully."

        ELSE

            PRINT "Error: Failed to retrieve data from Premier League API."

        END IF
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

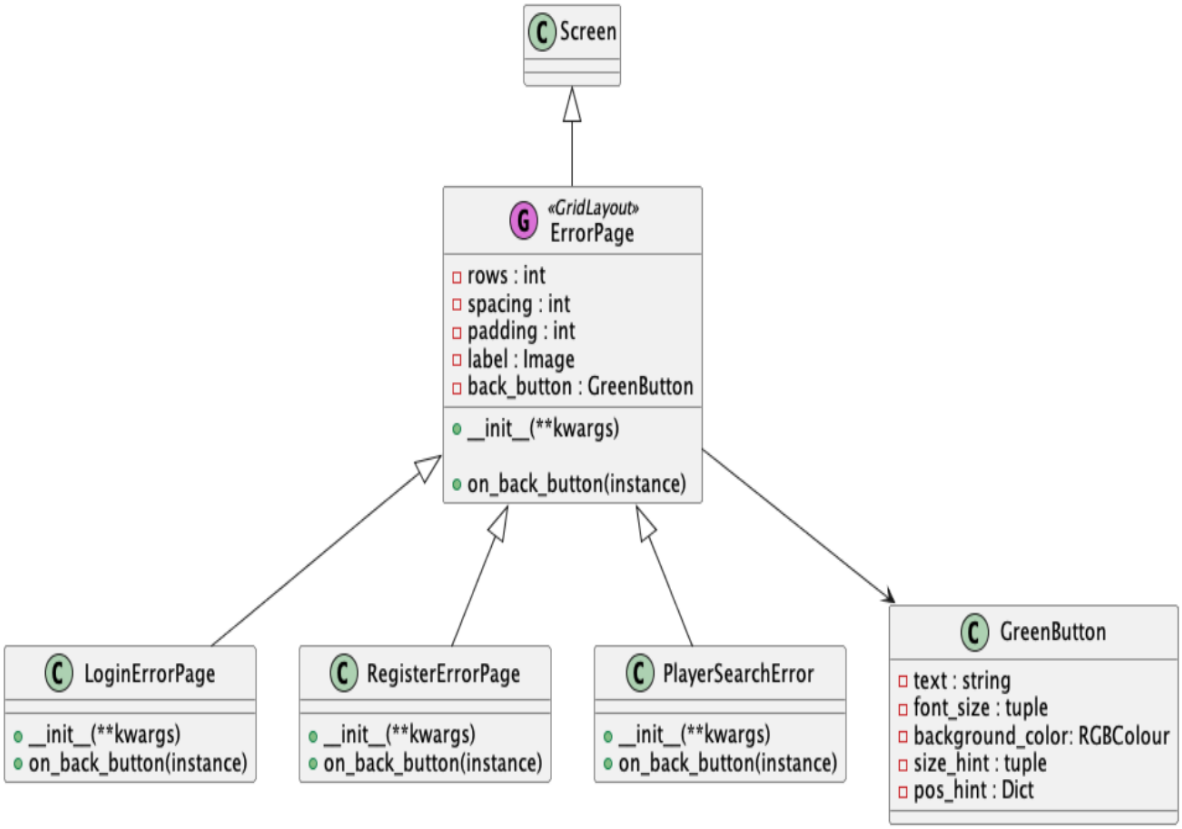
UML DIAGRAMS

As my client requires a GUI, and as I mention in my {Analysis}, I want the GUI to be easy to use and also look good. I will be basing my GUI's colour scheme on the Premier League app colours. Therefore, I will be using green- and purple-coloured buttons. I won't use the exact colour shades as I don't want my app to look like a part of an already existing app, however, I will keep them similar so I can keep the essence of what my client is used to in my app. For this I needed to design an efficient way of implementing classes of

buttons, which I can easily call throughout my code. This is because I will have a lot of these buttons all over my GUI and therefore this will make my implantation much more efficient and tidier. This GUI planning is all part of OBJECTIVE 4 {Analysis}.



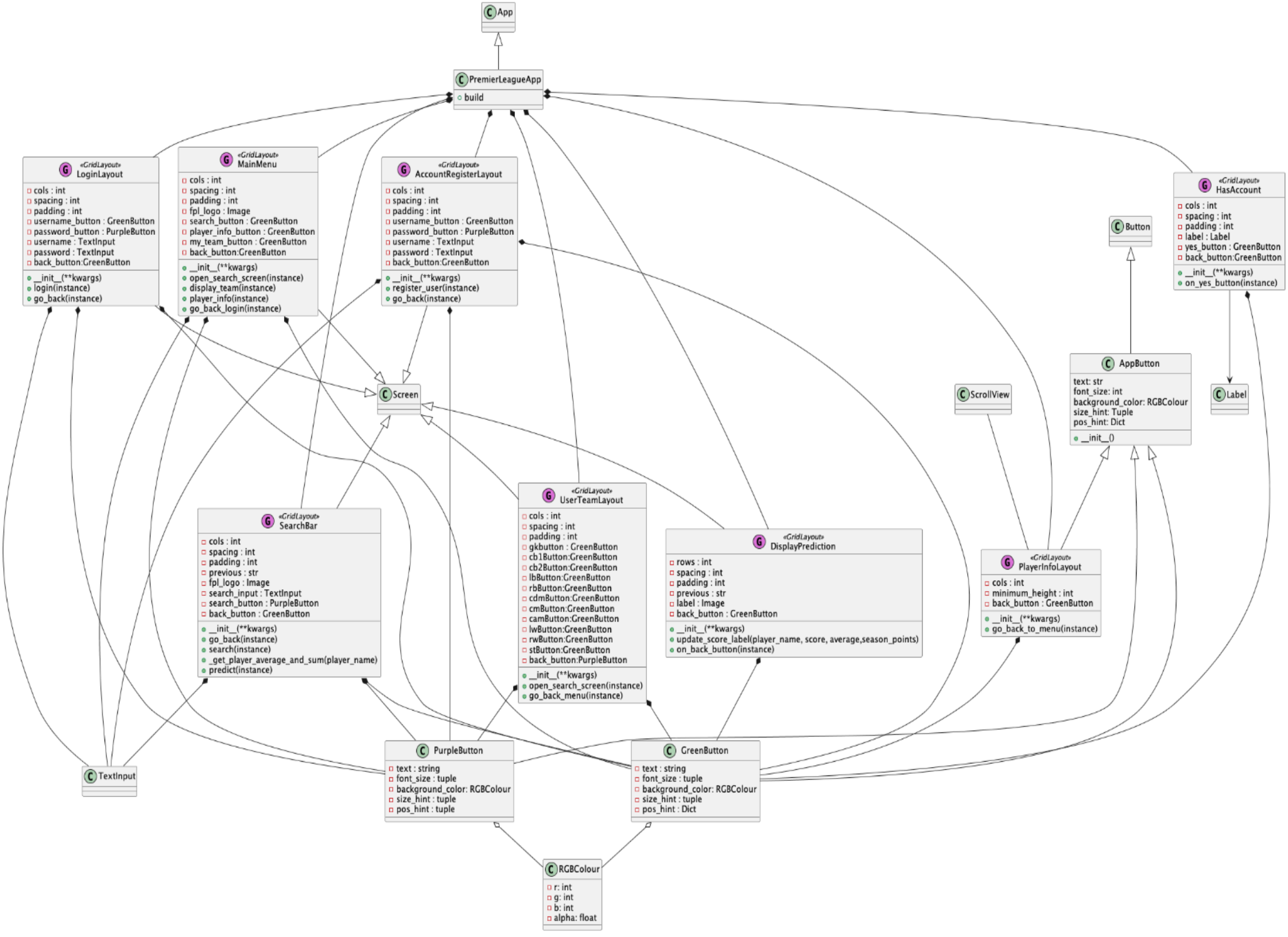
Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603



As my client wants a functional GUI as stated in OBJECTIVE 4 {Analysis}, I cannot afford to make my code, so it is not robust. There is a lot of opportunities for the user to input something which means that I will have to make my code robust to ensure that it cannot be broken. I want to do this so even if my client inputs something which isn't an acceptable input, my program will interact with him through a GUI. As there will be a lot of different error pages for different sorts of misinputs, I have planned to make a parent class with all the attributes that an error page will have, from which I can easily make different error pages with different messages and so that each page returns to the correct page after the user presses the back button. Above you can see how I have planned this in the form of another UML class relation diagram.

On the next page, you can see the relationships between all the classes that are related in a UML diagram like above.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603


App
PremierLeagueApp
build
«GridLayout»
LoginLayout
cols : int
spacing : int
padding : int
username_button : GreenButton
password_button : PurpleButton
username : TextInput
password : TextInput
back_button:GreenButton
__init__(**kwargs)
login(instance)
go_back(instance)
«GridLayout»
MainMenu
cols : int
spacing : int
padding : int
fpl_logo : Image
search_button : GreenButton
player_info_button : GreenButton
my_team_button : GreenButton
back_button:GreenButton
__init__(**kwargs)
open_search_screen(instance)
display_team(instance)
player_info(instance)
go_back_login(instance)
«GridLayout»
AccountRegisterLayout
cols : int
spacing : int
padding : int
username_button : GreenButton
password_button : PurpleButton
username : TextInput
password : TextInput
back_button:GreenButton
__init__(**kwargs)
register_user(instance)
go_back(instance)
«GridLayout»
HasAccount
cols : int
spacing : int
padding : int
label : Label
yes_button : GreenButton
back_button:GreenButton
__init__(**kwargs)
on_yes_button(instance)
Button
AppButton
text: str
font_size: int
background_color: RGBColour
size_hint: Tuple
pos_hint: Dict
__init__()
Label
ScrollView
Screen
«GridLayout»
SearchBar
cols : int
spacing : int
padding : int
previous : str
fpl_logo : Image
search_input : TextInput
search_button : PurpleButton
back_button : GreenButton
__init__(**kwargs)
go_back(instance)
search(instance)
_get_player_average_and_sum(player_name)
predict(instance)
«GridLayout»
UserTeamLayout
cols : int
spacing : int
padding : int
gkbutton : GreenButton
cb1Button:GreenButton
cb2Button:GreenButton
lbButton:GreenButton
rbButton:GreenButton
cdmButton:GreenButton
cmButton:GreenButton
camButton:GreenButton
lwButton:GreenButton
rwButton:GreenButton
stButton:GreenButton
back_button:PurpleButton
__init__(**kwargs)
open_search_screen(instance)
go_back_menu(instance)
«GridLayout»
DisplayPrediction
rows : int
spacing : int
padding : int
previous : str
label : Image
back_button : GreenButton
__init__(**kwargs)
update_score_label(player_name, score, average,season_points)
on_back_button(instance)
«GridLayout»
PlayerInfoLayout
cols : int
minimum_height : int
back_button : GreenButton
__init__(**kwargs)
go_back_to_menu(instance)
TextInput
PurpleButton
text : string
font_size : tuple
background_color: RGBColour
size_hint : tuple
pos_hint : tuple
GreenButton
text : string
font_size : tuple
background_color: RGBColour
size_hint : tuple
pos_hint : Dict
RGBColour
r: int
g: int
b: int
alpha: float

HARDWARE REQUIREMENTS

To be able to run my app on a device, certain hardware requirements are needed:

1 GHz 32-bit (x86) or 64-bit (x64) CPU

1 GB of RAM

128 MB of Graphics Memory

This apps usage will have around 30 MB RAM usage

APP FLOWCHART

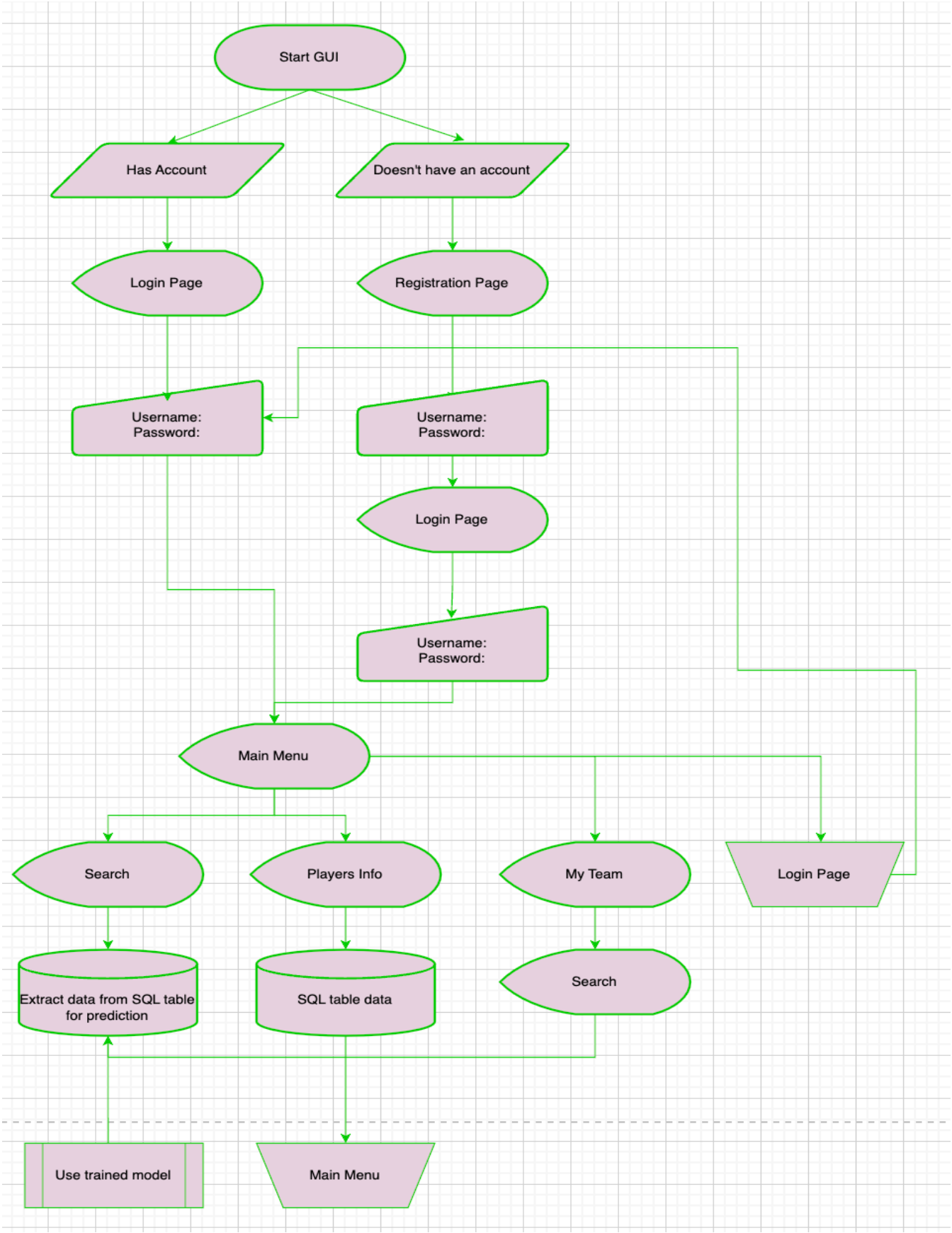
On the previous page, you see an elaborate UML diagram, this shows the inheritance, aggregation, and composition between different class. You can see an in-depth plan of how all of my GUI classes will work, this will once again allow me to complete OBJECTIVE 4 {Analysis} efficiently and a well-planned layout like the one above will mean I can implement my code with far less problems than I would have otherwise had to face.

Once I make my GUI, I need it to work seamlessly as a whole entity, and to do this I need to decide what screen will appear after each screen and how my client may use the GUI.

The below flowchart also shows the function of OBJECTIVE 3 {Analysis}, which is the account registration. As you can see, the app begins through an account login or registration system, and I can use this feature as a key foundation of any updates my client may want in the app. This flowchart below also mentions OBJECTIVE 2 {Analysis}, which is my SQL usage. In order to make a prediction using OBJECTIVE 5 {Analysis}, which is my machine learning model, I need to extract data about the player my client has searched up from the current game week and give that to the model to predict from.

You can see this flowchart on the next page:

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

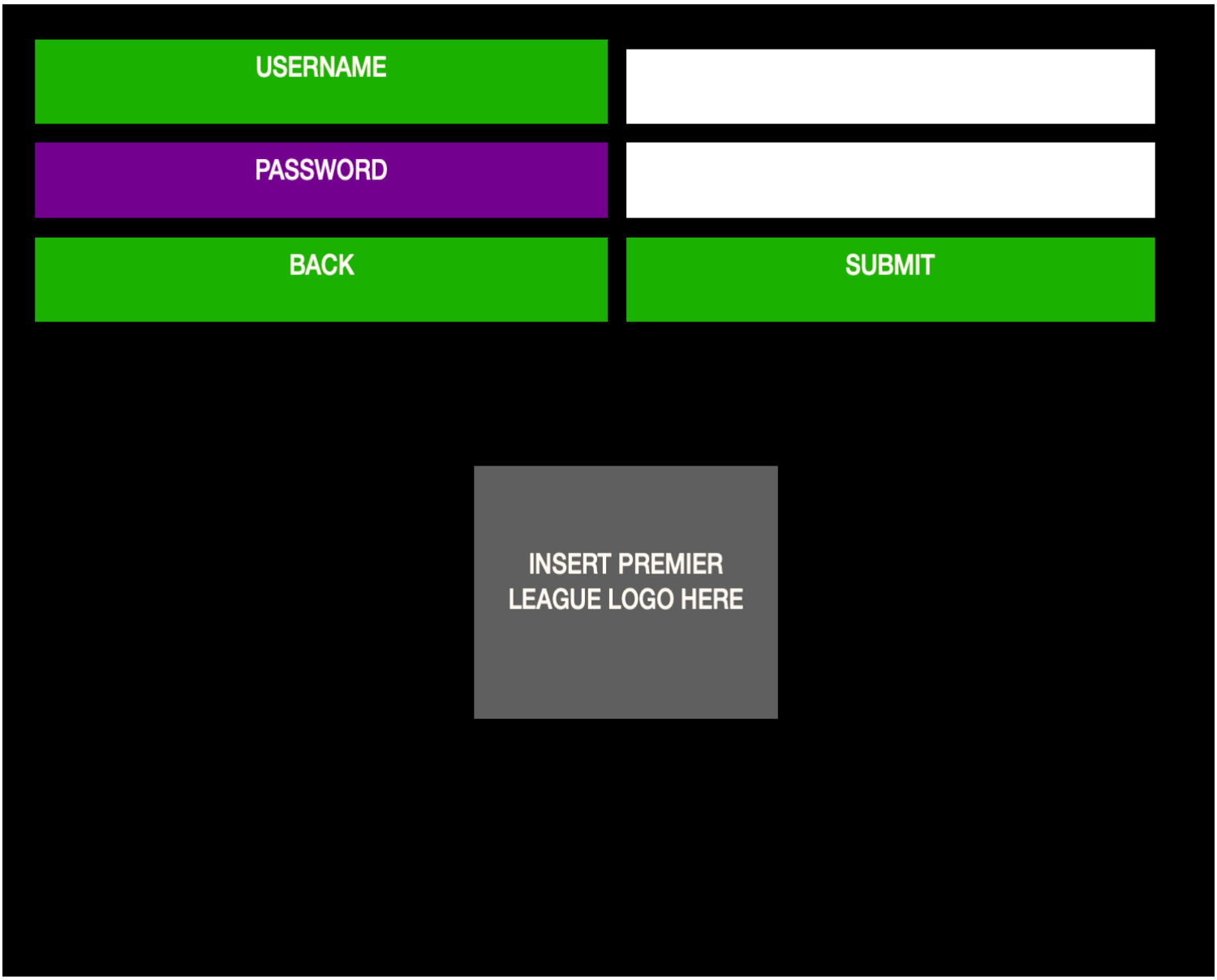
GUI DESIGN

To make a successful app, my GUI's design is very important. As mentioned in OBJECTIVE 4 {Analysis}, my client wishes for a simplistic and appealing GUI, which will allow him to easily ask for predictions and view the points information for all the players in the league. I have also added some additional screens as a way of making the GUI more relatable with his actual FPL team (like the 'My Team' screen on you will see in one of the designs for the GUI). I intend on going for a simplistic yet easy to use GUI, which will look good but also allow users to navigate through the app with ease. I have designed my different pages below:

Has Account?

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Login Page (Registration Page will be same but it will say Register instead of Submit)

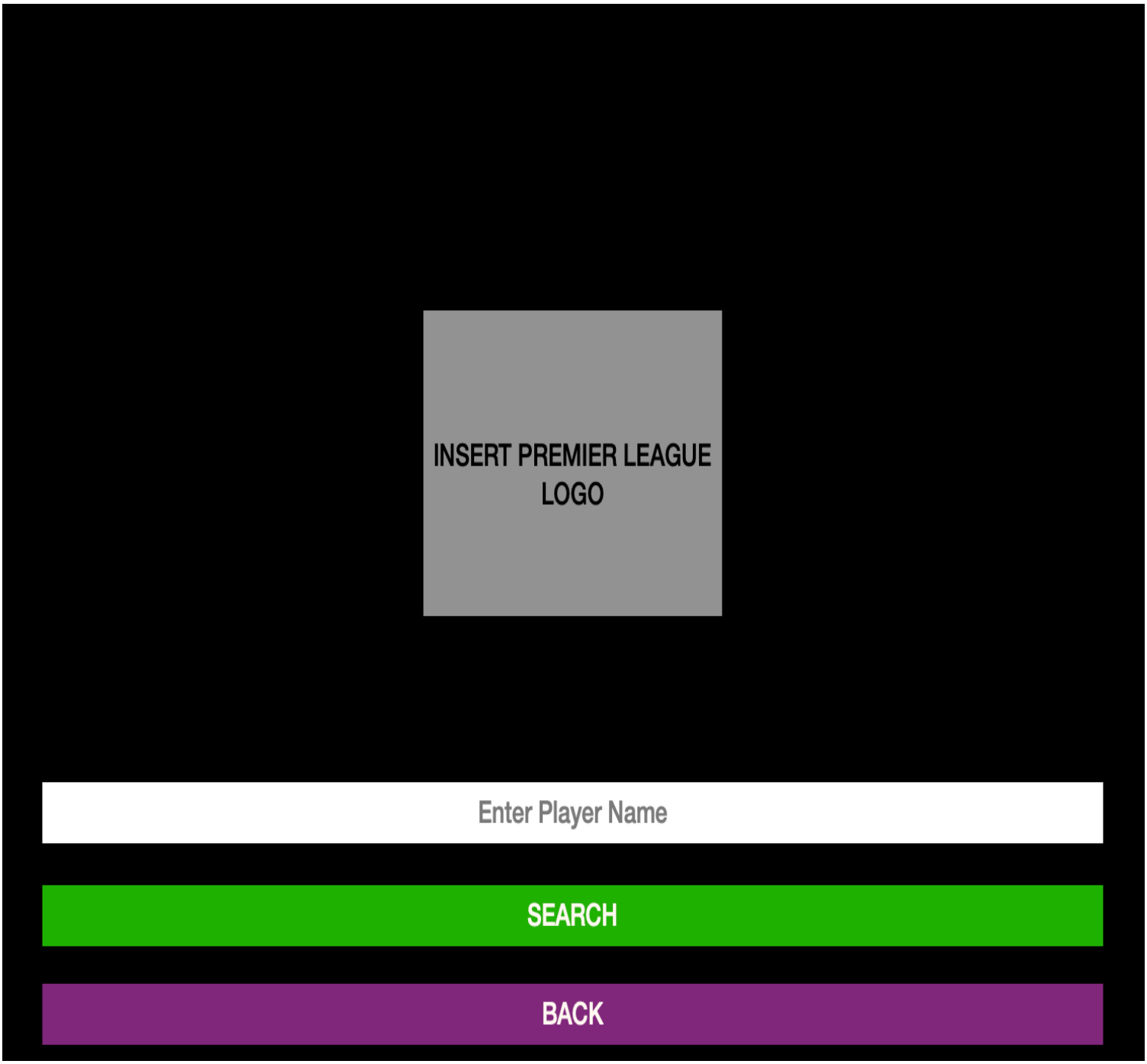


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Main Menu

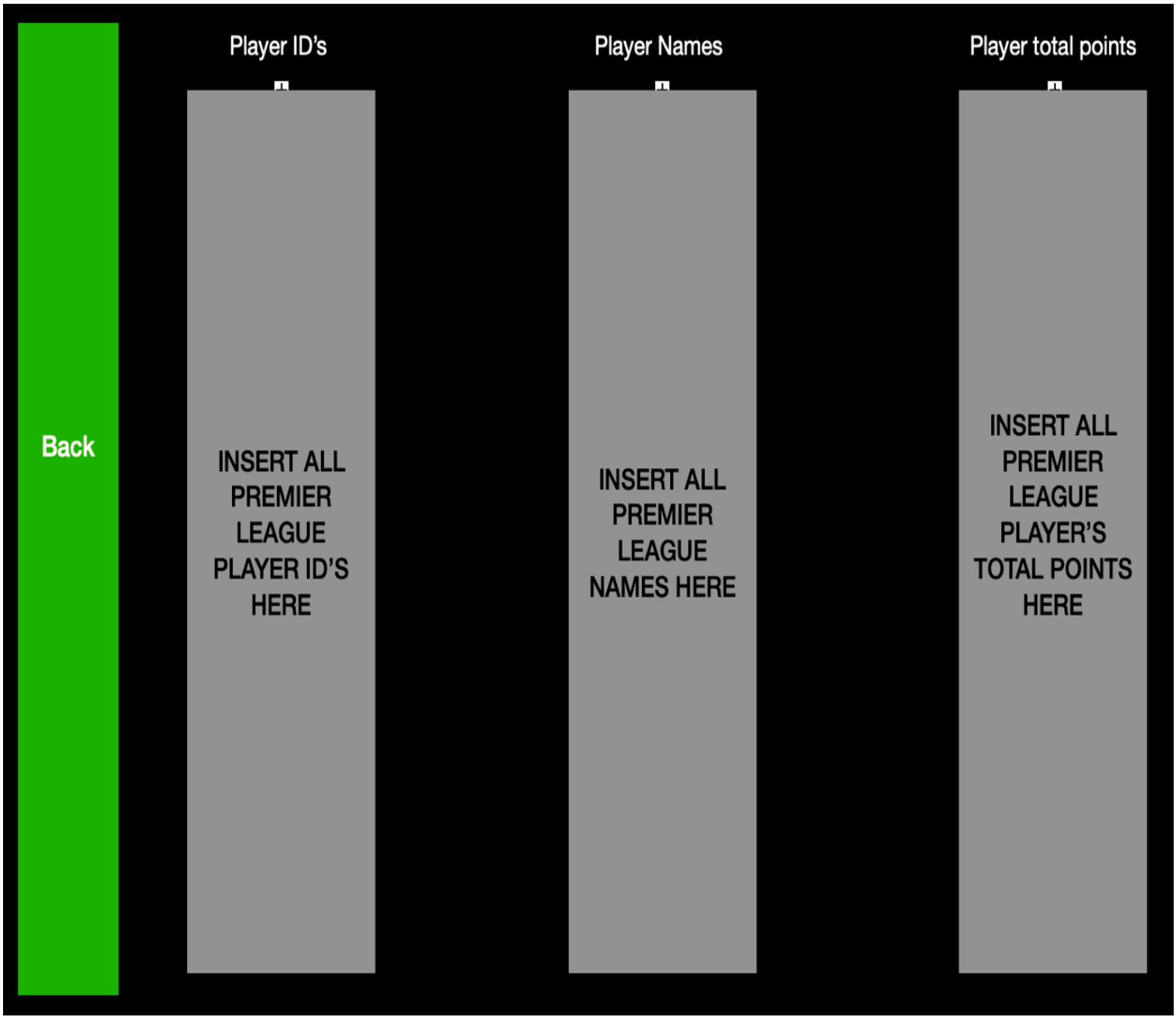
Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Search Bar



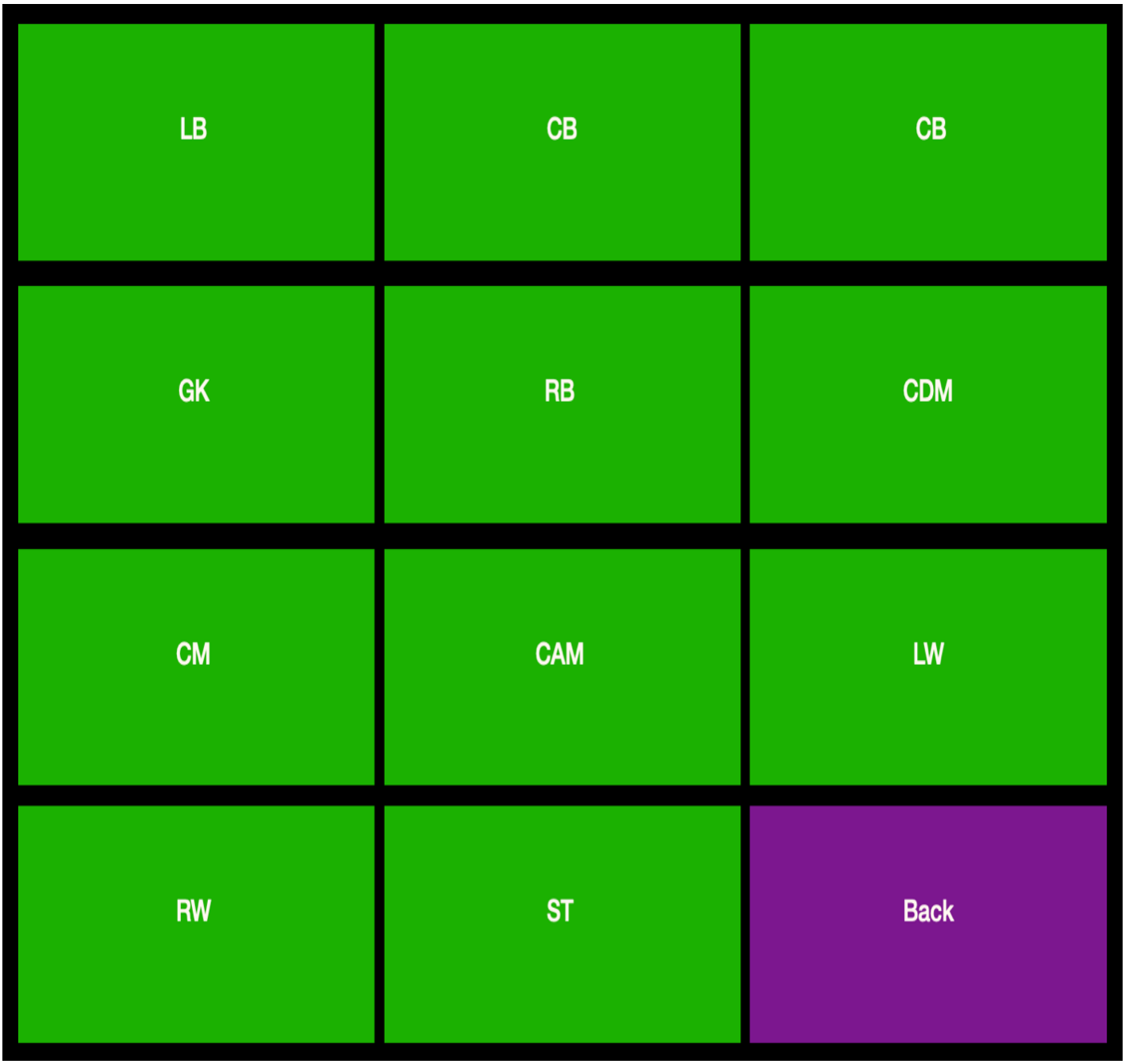
Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Players Info



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

My Team



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Database examples

These database tables are all an example of OBJECTICVE 2 {Analysis}, and they show how I will use SQL to manage my data. These tables aren't the only tables I will have but are an example. As you can see, below, I show the design of my user information table, which is 3NF normalised.

Field	*Key*	*Data Type*	*Validation*	*Notes*
Username	*Primary*	***String***		*This is protected from SQL injections using parameterized SQL*
Password		***Hash***	*Must be >5 chars.*	*Protected using hashing and encryption*

(3NF)

In addition to this, you can also see, I have shown the design for a table which will contain information about the footballers' games and their stats. This is also 3NF and has two primary keys, the id of the player and the game week. This data is collected from the Premier League API, and this is part of OBJECTIVE 1 {Analysis} and the data collected is then used for my OBJECTIVE 2 {Analysis}.

Field	*Key*	*Data Type*	*Notes*
id_player	*Primary*	*Int*	*All from premier League API.*
opponent_team		*Int*	
was_home		*Int*	*Treated as bool: 0/1*
round	*Primary*	*Int*	
transfers_in		*Int*	
selected		*Int*	
total_points		*Int*	

(3NF)

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Neural Network

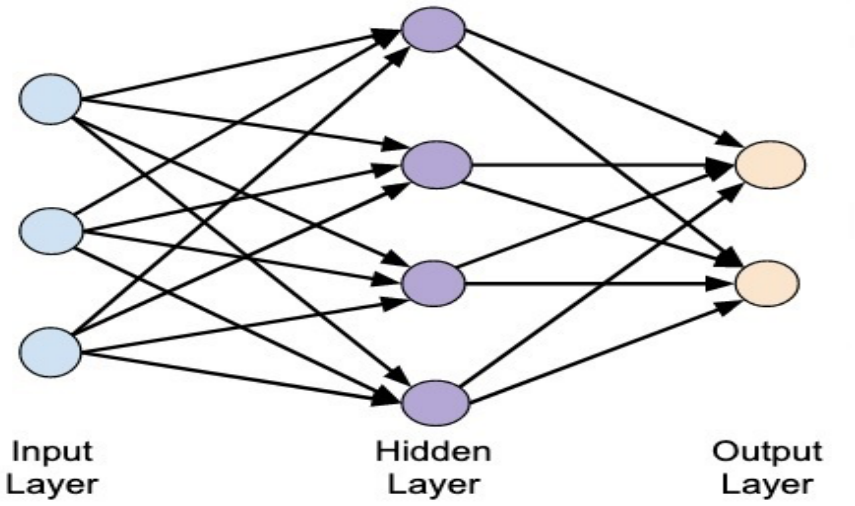
A neural network is what makes OBJECTIVE 5 {Analysis} come true, I need to make a neural network which will gets trained using a lot of data so I can have a machine learning model from which I can make player points predictions. As mentioned in OBJECTIVE 5 {Analysis}, once I make the main code for my neural network with the help of TensorFlow and additional research[8], I will have to train my model several times without saving it and tweak the 'epoch' and 'batch-size' values; I will most likely also have to change the number of layers I have in my model from 2 to 3 so my model can become more accurate.

What does the above actually mean?

A neural network can have two layers, the first layer being the input layer. This layer simply passes the data into the next layer. Then, you can go straight to the third layer, which is the output layer, this usually has a different activation function to the rest so it can give an output in an acceptable range. However, it is usually better to have a second layer in between the two layers previously mentioned; this layer is called the hidden layer, this is used with activation functions like ReLU and is used to train data based on non-linear results. As I mention in OBJECTIVE 5 {Analysis} and {Testing}, I believe I will create a better model if I use three layers instead of just two, and the activation functions I will use are ReLU for the first two layers, and the linear activation function for the output layer.

In addition to this, I will also have to tweak my epoch value; this is the number of times the neural network goes through the dataset and learns from it. The batch-size is the number of training examples which are used as dataset in one epoch, these are taken from the whole dataset.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603



Pseudocode for neural network using Keras TensorFlow

All part of OBJECTIVE 5 {Analysis} still, as this is my machine learning model pseudocode. It is not language specific and is just a general pseudocode but in reality, I would do this using the Keras and TensorFlow libraries.

```
# Initialize the neural network architecture
num_inputs ← [number of input features]
num_hidden_layers  ← [number of hidden layers]
num_neurons_per_layer  ← [list of number of neurons per hidden
layer]
num_outputs  ←  [number of output classes]

# Initialize weights and biases for each layer
weights ←  [list of weight matrices for each layer]
biases  ←  [list of bias vectors for each layer]

# Define the activation function for each layer
activations  ←  [list of activation functions for each layer]

# Define the loss function and optimization algorithm
loss_function  ← [loss function]
optimizer ← [optimization algorithm]

# Train the neural network

for epoch in range(num_epochs):
    # Shuffle the training data
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

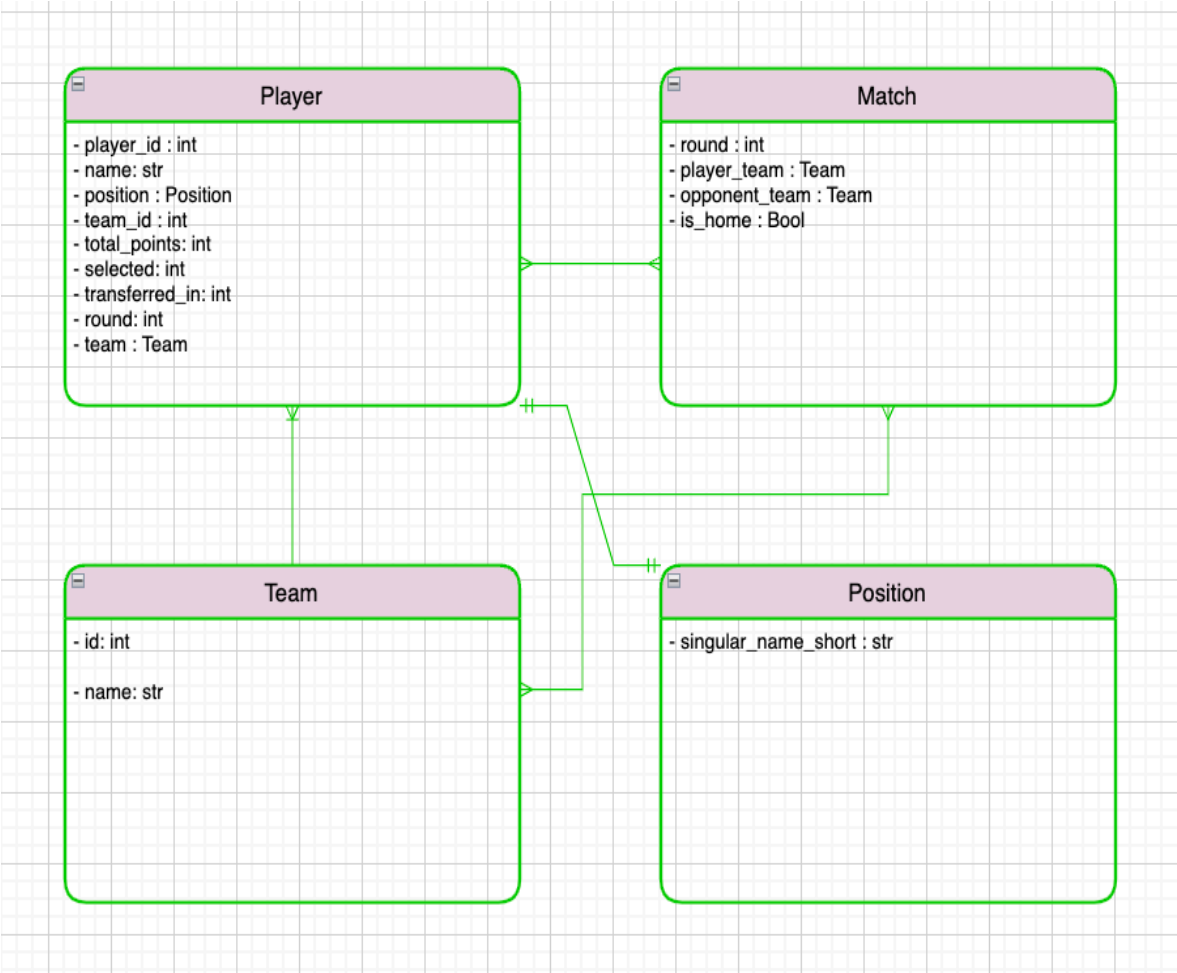
```
     shuffle(training_data)

    # Loop over training data in batches
    for batch_start in range(0, len(training_data), batch_size):

 # Extract input and output for the current batch

        batch_input ← 
training_data[batch_start:batch_start+batch_size, :-1]
        batch_output ← 
training_data[batch_start:batch_start+batch_size, -1]

        # Forward pass through the network
        hidden_layer_input ← dot(batch_input, weights[0]) + 
biases[0]

#dot means dot product

        hidden_layer_output ← 
activations[0](hidden_layer_input)

        for i in range(1, num_hidden_layers):

            layer_input ← dot(hidden_layer_output, weights[i]) 
+ biases[i]
            hidden_layer_output ← activations[i](layer_input)
        output_layer_input ← dot(hidden_layer_output, 
weights[-1]) + biases[-1]
        output_layer_output ← activations[-
1](output_layer_input)
        # Calculate the loss and gradients using backpropagation
        loss ← loss_function(batch_output, 
output_layer_output)
        gradients ← calculate_gradients(loss, weights, 
biases, activations)
        # Update the weights and biases using the optimization 
algorithm
        weights, biases ← optimizer(weights, biases, 
gradients)
    # Evaluate the performance on the validation set
    validation_loss ← evaluate_performance(validation_data, 
weights, biases, activations)

OUTPUT "Validation Loss: " + validation_loss
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Entity Relations

My main entities are my data. Hence the relationships will be between the players, their teams, and games etc. This is a representation of the relationships of my data entities, which is OBJECTIVE 1 {Analysis}. These will be related to each other and will be shown through my SQL tables : OBJECTIVE 2 {Analysis}

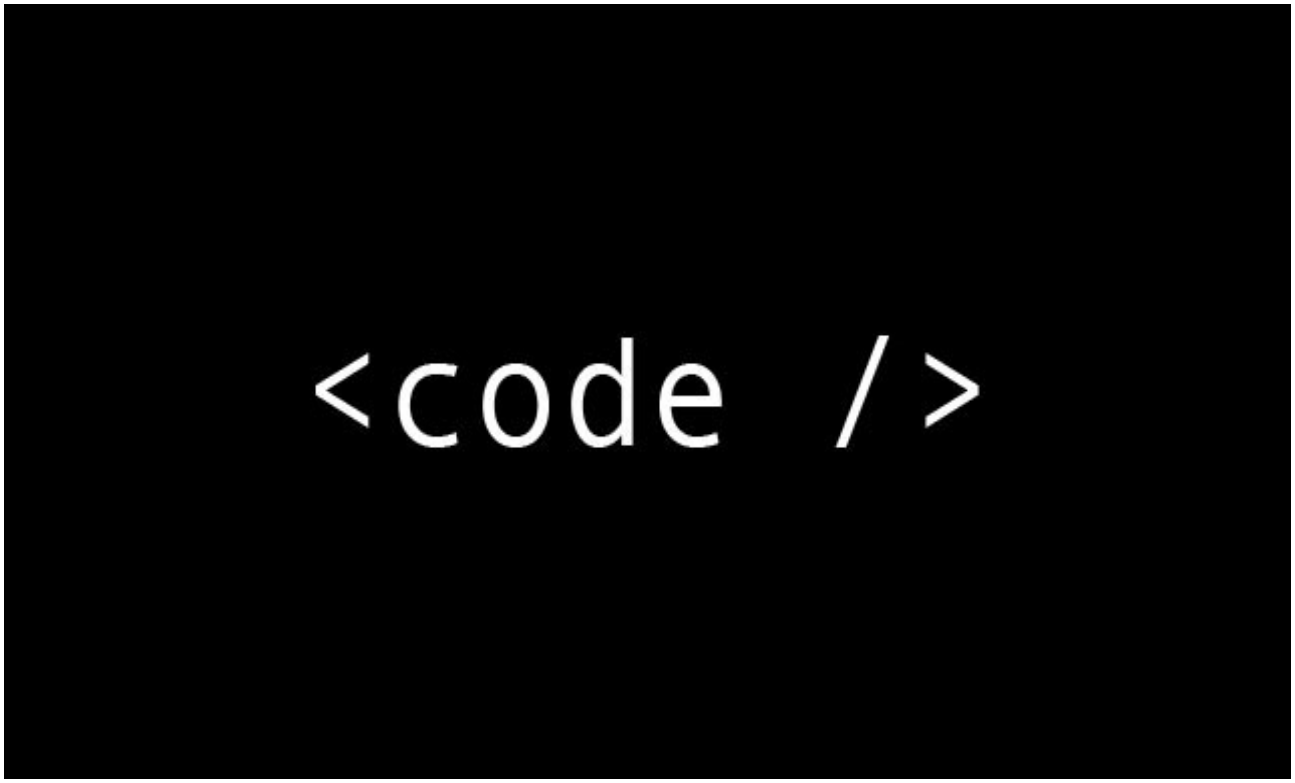
As you can see above there are many-to-many, one-to-one and one-to-many relationships in the above entity relationship diagram



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Technical Solution

Prannvat Singh



My technical solution below is split into three different scripts:

data.py **-** Module that handles retrieving all the data from the FPL API.

training_model.py **–** Module that handles the training of the neural network

prem_app.py **–** Module that handles the whole GUI

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

data.py

```
"""Module that handles retrieving all the data from the FPL API."""

import requests
import pandas as pd
import numpy as np
import csv
import sqlite3


BASE_URL = "https://fantasy.premierleague.com/api/"  # the base url for all of the FPL endpoints


def get_all_gameweek_data_for(player_code: int):
    """Gets all gameweek data for season for every player:"""
    # Getting gameweek information for specific players
    element_summary_URL = requests.get(
        BASE_URL + "element-summary/" + str(player_code) + "/"
    ).json()
    # ^^sending GET request to https://fantasy.premierleague.com/api/element-summary/{PID}/
    fixtures_df = pd.json_normalize(element_summary_URL["fixtures"])
    fixtures_df[["difficulty"]]
    specific_player_df = pd.json_normalize(element_summary_URL["history"])

    specific_player_df = pd.merge(
        specific_player_df,
        fixtures_df[["difficulty", "is_home"]],
        left_on="round",
        right_on="difficulty",
    )

    specific_player_df = specific_player_df[["difficulty"]]
    return specific_player_df


def past_season_gameweek_info(user_choice_player):
    '''Previous fixtures and results for players'''
    element_summary_url = requests.get(
        BASE_URL + "element-summary/" + str(user_choice_player) + "/"
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    ).json()

    # extract 'history' data from response into dataframe
    specific_player_df =
pd.json_normalize(element_summary_url["history"])

    return specific_player_df


def get_per_season_stats_for(player_code: int):
    """Get sum stats for past seasons players have played in the
league:"""
    # Getting gameweek information for specific players
    request_URL = requests.get(
        BASE_URL + "element-summary/" + str(player_code) + "/"
    ).json()
    # ^^sending GET request to
https://fantasy.premierleague.com/api/element-summary/{PID}/

    specific_player_df_history =
pd.json_normalize(request_URL["history_past"])
    return specific_player_df_history


def get_all_players_stats_for_this_season():
    """Makes a table with all the stats of all players in each
gameweek seperately"""
    request_URL = requests.get(BASE_URL + "bootstrap-
static/").json()
    # request_URL =
requests.get(BASE_URL+'event/'+str(gameweek)+'/live/').json()
    # I can now get player data usimg the 'elements' field of
the API:
    player_info = request_URL["elements"]
    # using Pandas, I can now make the data I have be in a
tabular format.
    # Pandas is a library made for exactly this purpose.
    pd.set_option("display.max_columns", None)
    # now I can create dataframe using the players info
    player_info = pd.json_normalize(request_URL["elements"])

    # creating dataframe for teams:
    teams_info = pd.json_normalize(request_URL["teams"])
    # player positions using the 'element-types' field:
    player_positions =
pd.json_normalize(request_URL["element_types"])
    # creating a points table which will contain all the info
for all
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    player_info = player_info[
        ["id", "first_name", "second_name", "web_name", "team",
"element_type"]
    ]
    # i will join the teamn names to the table
    player_info = player_info.merge(
        teams_info[["id", "name"]],
        left_on="team",
        right_on="id",
        suffixes=["_player", None],
    ).drop(["team", "id"], axis=1)
    # joining player positions:
    player_info = player_info.merge(
        player_positions[["id", "singular_name_short"]],
        left_on="element_type",
        right_on="id",
    ).drop(["element_type", "id"], axis=1)
    # next, i will use the progress_apply() dataframe method,
    # that comes with pandas,
    #  to apply past_season_gameweek_info() function
    # to every row in players_info dataframe.
    # getting gameweek histories for each player

    from tqdm.auto import tqdm

    tqdm.pandas()
    player_points =
player_info["id_player"].progress_apply(past_season_gameweek_inf
o)
    # combining results into one dataframe
    player_points = pd.concat(df for df in player_points)
    player_points = player_info[["id_player", "web_name",
"singular_name_short"]].merge(
        player_points, left_on="id_player", right_on="element"
    )

    return player_points


def total_stats_sum_df():  # this gives the sum of everything
and orders the stats in most points scored.
    total_stats_season_df =
get_all_players_stats_for_this_season()
    total_stats_season_df = (
        total_stats_season_df.groupby(["element", "web_name",
"singular_name_short"])
        .agg(
            {
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                "total_points": "sum",
                "goals_scored": "sum",
                "assists": "sum",
                "goals_conceded": "sum",
            }
        )
        .reset_index()
        .sort_values("total_points", ascending=False)
    )
    writer = pd.ExcelWriter("player_points.xlsx")
    total_stats_season_df.to_excel(writer)
    writer.save()
    writer = pd.ExcelWriter("player_points.xlsx")
    total_stats_season_df.to_excel(writer)
    writer.save()
    return total_stats_season_df


def get_dataset_to_train_nnetwork():

    training_df = get_all_players_stats_for_this_season()

    training_df = training_df[
        [
            "id_player",
            "opponent_team",
            "was_home",
            "round",
            "transfers_in",
            "selected",
            "total_points",
        ]
    ]
    training_df["was_home"] =
training_df["was_home"].astype(int)
    training_df.to_csv("testsql.csv", index=False)
    training_list = training_df.values.tolist()

    file = open("training.csv", "w")
    writer = csv.writer(file)
    writer.writerows(training_list)
    return training_df


def sql_player_database(df):

    """Using sqlite3 to make all sql tables. Uses all
techniques."""
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    # print(model_df)
    player_df = df[["element", "web_name",
"singular_name_short"]]
    players_stats_df = df[["element", "web_name",
"total_points"]]
    # print(players_stats_df)
    conn = sqlite3.connect("fpl_players_db.sqlite")
    cur = conn.cursor()
    # #Creating these tables on my sql database, they can then
be used for aggregate functions etc and cross-parameterized sql.
    fpl_player_table = """
                CREATE TABLE IF NOT EXISTS fpl_player_table (
                element INTEGER PRIMARY KEY,
                web_name TEXT,
                singular_name_short TEXT
                );
                """
    cur.execute(fpl_player_table)
    # INSERT OR IGNORE INTO only exerts if it doesn't already
exist in table.
    for _, row in player_df.iterrows():
        cur.execute(
            "INSERT OR IGNORE INTO fpl_player_table (element,
web_name, singular_name_short) VALUES (?,?,?)",
            (row["element"], row["web_name"],
row["singular_name_short"]),
        )
    conn.commit()

    fpl_player_stats_table = """
                CREATE TABLE IF NOT EXISTS
fpl_player_stats_table (
                element INTEGER PRIMARY KEY,
                web_name TEXT,
                total_points INTEGER
                );
                """
    cur.execute(fpl_player_stats_table)

    for _, row in players_stats_df.iterrows():
        cur.execute(
            "INSERT OR IGNORE INTO fpl_player_stats_table
(element, web_name, total_points) VALUES (?,?,?)",
            (row["element"], row["web_name"],
row["total_points"]),
        )
    conn.commit()
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    # Create the table
    cur.execute(
        """
    CREATE TABLE IF NOT EXISTS fpl_player_model_table (
    id_player INTEGER,
    opponent_team INTEGER,
    was_home INTEGER,
    round INTEGER,
    transfers_in INTEGER,
    selected INTEGER,
    total_points INTEGER,
    PRIMARY KEY (id_player, round)
    );
    """
    )

    # Load the CSV file into a pandas DataFrame
    df = pd.read_csv("testsql.csv")

    # Insert the data from the DataFrame into the table
    df.to_sql("fpl_player_model_table", conn,
if_exists="replace", index=False)

    conn.commit()

    query = """
    CREATE TABLE IF NOT EXISTS fpl_player_final_table AS
    SELECT *
    FROM fpl_player_model_table
    JOIN fpl_player_table
    ON fpl_player_model_table.id_player =
fpl_player_table.element"""

    try:
        cur.execute(query)
        conn.commit()
    except Exception as e:
        print("Error:", e)

    user_search = str(input("Enter player Name:"))
    # CROSS-PARAMETERIZED SQL ASWELL AS AGGREGATE FUNCTIONS.
    query = """
    SELECT AVG(fpl_player_stats_table.total_points)
    FROM fpl_player_table
    INNER JOIN fpl_player_stats_table
    ON fpl_player_table.element = fpl_player_stats_table.element
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    WHERE fpl_player_table.web_name = ? AND
fpl_player_stats_table.web_name = ?
    """

    # '?' is used as a placeholder for values in SQL table.
Known as parameterized SQL.
    result = cur.execute(
        query, (user_search, user_search)
    ).fetchall()  # fetchall() retrieves all the rows returned
from SELECT statement
    print(result)

    conn.close()

    # Creating player database with their points

# Connect to the database

'''Below is all to get all the player info sorted in terms of
points for the GUI'''
conn = sqlite3.connect('fpl_players_db.sqlite')
c = conn.cursor()



# Define the merge sort function
def merge_sort(data, col):
    if len(data) <= 1:
        return data
    mid = len(data) // 2
    left = data[:mid]
    right = data[mid:]
    left = merge_sort(left, col)
    right = merge_sort(right, col)
    return merge(left, right, col)

def merge(left, right, col):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i][col] >= right[j][col]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    return result


import heapq

def optimize_fpl_team(start_team, graph):
    # Set all nodes to unvisited and initialize weights to
infinity
    unvisited = set(graph.keys())
    weights = {node: float('inf') for node in graph.keys()}
    weights[start_team] = 0

    # Initialize the priority queue with the start node
    priorityQ = []
    heapq.heappush(priorityQ, (weights[start_team], start_team))

    while priorityQ:
        # Extract the node with the lowest weight
        curr_weight, curr_node = heapq.heappop(priorityQ)

        # Mark the node as visited
        unvisited.remove(curr_node)

        # Update the weights of unvisited neighbors
        for neighbor, weight in graph[curr_node].items():
            if neighbor in unvisited:
                new_weight = curr_weight + weight
                if new_weight < weights[neighbor]:
                    weights[neighbor] = new_weight
                    heapq.heappush(priorityQ, (new_weight,
neighbor))

    # Extract the optimized FPL team
    optimized_team = []
    curr_node = start_team
    while curr_node:
        optimized_team.append(curr_node)
        min_weight = float('inf')
        next_node = None
        for neighbor, weight in graph[curr_node].items():
            if weights[neighbor] < min_weight:
                min_weight = weights[neighbor]
                next_node = neighbor
        curr_node = next_node if next_node != start_team else
None

    return optimized_team
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
graph = {
    'Salah': {'Mane': 15, 'Alexander-Arnold': 10},
    'Mane': {'Salah': 15, 'Firmino': 5},
    'Alexander-Arnold': {'Salah': 10, 'Robertson': 8},
    'Robertson': {'Alexander-Arnold': 8, 'Firmino': 4},
    'Firmino': {'Mane': 5, 'Robertson': 4},
}


if __name__ == "__main__":

    df = total_stats_sum_df()
    # print(df)
    # print(model_df)

    sql_player_database(df)
    print(get_all_players_stats_for_this_season())

    conn = sqlite3.connect('fpl_players_db.sqlite')
    c = conn.cursor()
    # Call the merge sort function on the data
    # Retrieve the data from the table
    c.execute("SELECT * FROM fpl_player_stats_table")

    data = c.fetchall()
    sorted_data = merge_sort(data, col=2)


    # Create the new table
    c.execute("CREATE TABLE IF NOT EXISTS sorted_table (element
INTEGER PRIMARY KEY, web_name TEXT, total_points INTEGER)")


    # # Insert the sorted data into the new table
    for row in sorted_data:
        # import pdb; pdb.set_trace()
        print(row)

        c.execute("INSERT sorted_table (element, web_name,
total_points) VALUES (?, ?, ?)", (row))

    # # Commit the changes and close the connection
    conn.commit()
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Techniques I have used in above script with specific parts of code:

Cross-table parameterised SQL
Aggregate SQL Functions
User/CASE-generated DDL scripts:

```
def sql_player_database(df):

    """Using sqlite3 to make all sql tables. Uses all
techniques."""

    # print(model_df)
    player_df = df[["element", "web_name",
"singular_name_short"]]
    players_stats_df = df[["element", "web_name",
"total_points"]]
    # print(players_stats_df)
    conn = sqlite3.connect("fpl_players_db.sqlite")
    cur = conn.cursor()
    # #Creating these tables on my sql database, they can then
be used for aggregate functions etc and cross-parameterized sql.
    fpl_player_table = """
                CREATE TABLE IF NOT EXISTS fpl_player_table (
                element INTEGER PRIMARY KEY,
                web_name TEXT,
                singular_name_short TEXT
                );
                """
    cur.execute(fpl_player_table)
    # INSERT OR IGNORE INTO only exerts if it doesn't already
exist in table.
    for _, row in player_df.iterrows():
        cur.execute(
            "INSERT OR IGNORE INTO fpl_player_table (element,
web_name, singular_name_short) VALUES (?,?,?)",
            (row["element"], row["web_name"],
row["singular_name_short"]),
        )
    conn.commit()

    fpl_player_stats_table = """
                CREATE TABLE IF NOT EXISTS
fpl_player_stats_table (
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                element INTEGER PRIMARY KEY,
                web_name TEXT,
                total_points INTEGER
                );
                """
    cur.execute(fpl_player_stats_table)

    for _, row in players_stats_df.iterrows():
        cur.execute(
            "INSERT OR IGNORE INTO fpl_player_stats_table
(element, web_name, total_points) VALUES (?,?,?)",
            (row["element"], row["web_name"],
row["total_points"]),
        )
    conn.commit()


    # Create the table
    cur.execute(
        """
    CREATE TABLE IF NOT EXISTS fpl_player_model_table (
    id_player INTEGER,
    opponent_team INTEGER,
    was_home INTEGER,
    round INTEGER,
    transfers_in INTEGER,
    selected INTEGER,
    total_points INTEGER,
    PRIMARY KEY (id_player, round)
    );
    """
    )

    # Load the CSV file into a pandas DataFrame
    df = pd.read_csv("testsql.csv")

    # Insert the data from the DataFrame into the table
    df.to_sql("fpl_player_model_table", conn,
if_exists="replace", index=False)

    conn.commit()

    query = """
    CREATE TABLE IF NOT EXISTS fpl_player_final_table AS
    SELECT *
    FROM fpl_player_model_table
    JOIN fpl_player_table
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    ON fpl_player_model_table.id_player =
fpl_player_table.element"""

    try:
        cur.execute(query)
        conn.commit()
    except Exception as e:
        print("Error:", e)

    user_search = str(input("Enter player Name:"))
    # CROSS-PARAMETERIZED SQL ASWELL AS AGGREGATE FUNCTIONS.
    query = """
    SELECT AVG(fpl_player_stats_table.total_points)
    FROM fpl_player_table
    INNER JOIN fpl_player_stats_table
    ON fpl_player_table.element = fpl_player_stats_table.element
    WHERE fpl_player_table.web_name = ? AND
fpl_player_stats_table.web_name = ?
    """

    # '?' is used as a placeholder for values in SQL table.
Known as parameterized SQL.
    result = cur.execute(
        query, (user_search, user_search)
    ).fetchall()  # fetchall() retrieves all the rows returned
from SELECT statement
    print(result)

    conn.close()

    # Creating player database with their points

# Connect to the database

'''Below is all to get all the player info sorted in terms of
points for the GUI'''
conn = sqlite3.connect('fpl_players_db.sqlite')
c = conn.cursor()
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Graph/Tree Traversal
List Operations
Recursive algorithms
Stack/Queue Operations

```
import heapq

def optimize_fpl_team(start_team, graph):
    # Set all nodes to unvisited and initialize weights to
infinity
    unvisited = set(graph.keys())
    weights = {node: float('inf') for node in graph.keys()}
    weights[start_team] = 0

    # Initialize the priority queue with the start node
    priorityQ = []
    heapq.heappush(priorityQ, (weights[start_team], start_team))

    while priorityQ:
        # Extract the node with the lowest weight
        curr_weight, curr_node = heapq.heappop(priorityQ)

        # Mark the node as visited
        unvisited.remove(curr_node)

        # Update the weights of unvisited neighbors
        for neighbor, weight in graph[curr_node].items():
            if neighbor in unvisited:
                new_weight = curr_weight + weight
                if new_weight < weights[neighbor]:
                    weights[neighbor] = new_weight
                    heapq.heappush(priorityQ, (new_weight,
neighbor))

    # Extract the optimized FPL team
    optimized_team = []
    curr_node = start_team
    while curr_node:
        optimized_team.append(curr_node)
        min_weight = float('inf')
        next_node = None
        for neighbor, weight in graph[curr_node].items():
            if weights[neighbor] < min_weight:
                min_weight = weights[neighbor]
                next_node = neighbor
        curr_node = next_node if next_node != start_team else
None
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    return optimized_team

graph = {
    'Salah': {'Mane': 15, 'Alexander-Arnold': 10},
    'Mane': {'Salah': 15, 'Firmino': 5},
    'Alexander-Arnold': {'Salah': 10, 'Robertson': 8},
    'Robertson': {'Alexander-Arnold': 8, 'Firmino': 4},
    'Firmino': {'Mane': 5, 'Robertson': 4},
}
```

Mergesort or similarly efficient sort
Recursive algorithms
List Operations

```
# Define the merge sort function
def merge_sort(data, col):
    if len(data) <= 1:
        return data
    mid = len(data) // 2
    left = data[:mid]
    right = data[mid:]
    left = merge_sort(left, col)
    right = merge_sort(right, col)
    return merge(left, right, col)

def merge(left, right, col):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i][col] >= right[j][col]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Calling parameterised Web service API's and parsing JSON/XML to service a complex client-server model

```
BASE_URL = "https://fantasy.premierleague.com/api/"  # the base
url for all of the FPL endpoints


def get_all_gameweek_data_for(player_code: int):
    """Gets all gameweek data for season for every player:"""
    # Getting gameweek information for specific players
    element_summary_URL = requests.get(
        BASE_URL + "element-summary/" + str(player_code) + "/"
    ).json()
    # ^^sending GET request to
https://fantasy.premierleague.com/api/element-summary/{PID}/
    fixtures_df =
pd.json_normalize(element_summary_URL["fixtures"])
    fixtures_df[["difficulty"]]
    specific_player_df =
pd.json_normalize(element_summary_URL["history"])

    specific_player_df = pd.merge(
        specific_player_df,
        fixtures_df[["difficulty", "is_home"]],
        left_on="round",
        right_on="difficulty",
    )

    specific_player_df = specific_player_df[["difficulty"]]
    return specific_player_df


def past_season_gameweek_info(user_choice_player):
    '''Previous fixtures and results for players'''
    element_summary_url = requests.get(
        BASE_URL + "element-summary/" + str(user_choice_player)
+ "/"
    ).json()

    # extract 'history' data from response into dataframe
    specific_player_df =
pd.json_normalize(element_summary_url["history"])

    return specific_player_df
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
def get_per_season_stats_for(player_code: int):
    """Get sum stats for past seasons players have played in the
league:"""
    # Getting gameweek information for specific players
    request_URL = requests.get(
        BASE_URL + "element-summary/" + str(player_code) + "/"
    ).json()
    # ^^sending GET request to
https://fantasy.premierleague.com/api/element-summary/{PID}/

    specific_player_df_history =
pd.json_normalize(request_URL["history_past"])
    return specific_player_df_history


def get_all_players_stats_for_this_season():
    """Makes a table with all the stats of all players in each
gameweek seperately"""
    request_URL = requests.get(BASE_URL + "bootstrap-
static/").json()
    # request_URL =
requests.get(BASE_URL+'event/'+str(gameweek)+'/live/').json()
    # I can now get player data usimg the 'elements' field of
the API:
    player_info = request_URL["elements"]
    # using Pandas, I can now make the data I have be in a
tabular format.
    # Pandas is a library made for exactly this purpose.
    pd.set_option("display.max_columns", None)
    # now I can create dataframe using the players info
    player_info = pd.json_normalize(request_URL["elements"])

    # creating dataframe for teams:
    teams_info = pd.json_normalize(request_URL["teams"])
    # player positions using the 'element-types' field:
    player_positions =
pd.json_normalize(request_URL["element_types"])
    # creating a points table which will contain all the info
for all
    player_info = player_info[
        ["id", "first_name", "second_name", "web_name", "team",
"element_type"]
    ]
    # i will join the teamn names to the table
    player_info = player_info.merge(
        teams_info[["id", "name"]],
        left_on="team",
        right_on="id",
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        suffixes=["_player", None],
    ).drop(["team", "id"], axis=1)
    # joining player positions:
    player_info = player_info.merge(
        player_positions[["id", "singular_name_short"]],
        left_on="element_type",
        right_on="id",
    ).drop(["element_type", "id"], axis=1)
    # next, i will use the progress_apply() dataframe method,
    # that comes with pandas,
    #  to apply past_season_gameweek_info() function
    # to every row in players_info dataframe.
    # getting gameweek histories for each player

    from tqdm.auto import tqdm

    tqdm.pandas()
    player_points =
player_info["id_player"].progress_apply(past_season_gameweek_inf
o)
    # combining results into one dataframe
    player_points = pd.concat(df for df in player_points)
    player_points = player_info[["id_player", "web_name",
"singular_name_short"]].merge(
        player_points, left_on="id_player", right_on="element"
    )

    return player_points


def total_stats_sum_df():  # this gives the sum of everything
and orders the stats in most points scored.
    total_stats_season_df =
get_all_players_stats_for_this_season()
    total_stats_season_df = (
        total_stats_season_df.groupby(["element", "web_name",
"singular_name_short"])
        .agg(
            {
                "total_points": "sum",
                "goals_scored": "sum",
                "assists": "sum",
                "goals_conceded": "sum",
            }
        )
        .reset_index()
        .sort_values("total_points", ascending=False)
    )
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    writer = pd.ExcelWriter("player_points.xlsx")
    total_stats_season_df.to_excel(writer)
    writer.save()
    writer = pd.ExcelWriter("player_points.xlsx")
    total_stats_season_df.to_excel(writer)
    writer.save()
    return total_stats_season_df


def get_dataset_to_train_nnetwork():

    training_df = get_all_players_stats_for_this_season()

    training_df = training_df[
        [
            "id_player",
            "opponent_team",
            "was_home",
            "round",
            "transfers_in",
            "selected",
            "total_points",
        ]
    ]
    training_df["was_home"] =
training_df["was_home"].astype(int)
    training_df.to_csv("testsql.csv", index=False)
    training_list = training_df.values.tolist()

    file = open("training.csv", "w")
    writer = csv.writer(file)
    writer.writerows(training_list)
    return training_df
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

training_model.py

```
"""Module that handles all the Neural Network training."""
import tensorflow
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from numpy import loadtxt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.models import load_model

def train_model():
    """Train the keras model"""

    data = loadtxt("training.csv", delimiter=",")
    # Load data from CSV file
    # Split data into input and output
    X = data[:, :6]
    y = data[:, 6]

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

    # Scale the input data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create a neural network with three layers
    model = Sequential()
    model.add(Dense(12, input_dim=6, activation="relu"))
    model.add(Dense(10, activation="relu"))
    model.add(Dense(1, activation="linear"))

    # Compile the model
    model.compile(loss="mean_squared_error", optimizer="adam")

    # Train the model
    model.fit(X_train, y_train, epochs=150, batch_size=34, )

    # Evaluate the model on the test data
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    test_loss = model.evaluate(X_test, y_test, verbose=0)
    print("Test loss:", test_loss)
    model.save('fpl_predictor.h5')
    return X
if __name__ == '__main__':
   train_model(
```

Techniques I have used in above script with specific parts of code:

List operations

```
def train_model():
    """Train the keras model"""

    data = loadtxt("training.csv", delimiter=",")

    # Load data from CSV file

    # Split data into input and output
    X = data[:, :6]
    y = data[:, 6
```

Complex user-defined algorithms (e.g., optimisation, minimisation, scheduling, patter matching) or equivalent difficulty

```
    # Scale the input data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create a neural network with three layers
    model = Sequential()
    model.add(Dense(12, input_dim=6, activation="relu"))
    model.add(Dense(10, activation="relu"))
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    model.add(Dense(1, activation="linear"))

    # Compile the model
    model.compile(loss="mean_squared_error", optimizer="adam")

    # Train the model
    model.fit(X_train, y_train, epochs=150, batch_size=34, )

    # Evaluate the model on the test data
    test_loss = model.evaluate(X_test, y_test, verbose=0)
    print("Test loss:", test_loss)
    model.save('fpl_predictor.h5')
    return X
```

app.py

```
"""Module that handles all the GUI"""
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.image import Image
from kivy.uix.label import Label
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.textinput import TextInput
from typing import Tuple, Dict
import sqlite3
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.screenmanager import Screen
from kivy.uix.scrollview import ScrollView
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow
import hashlib

class RGBColour:
    def __init__(self, r: int, g: int, b: int, a: float = 1):
        self.r: int = r
        self.g: int = g
        self.b: int = b
        self.alpha: float = a


class AppButton(Button):
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    '''making a parent class from which I can have child buttons
easily'''
    def __init__(
        self,
        text: str,
        font_size: int,
        background_color: RGBColour,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        #kivy has rgba values which are the actual rgb values /
255
        background_color: Tuple = (
            background_color.r / 255,
            background_color.g / 255,
            background_color.b / 255,
            background_color.alpha,
        )
        super().__init__(
            #all of below will be attributes I will vary in
buttons.
            text=text,
            font_size=font_size,
            background_color=background_color,
            size_hint=size_hint,
            pos_hint=pos_hint,
            **kwargs,
        )

class GreenButton(AppButton):
    '''Inheritance from parent class AppButton'''
    def __init__(
        self,
        text: str,
        font_size: int,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        super().__init__(
            text=text,
            font_size=font_size,
            #setting RGBA values
            background_color=RGBColour(46, 204, 113,1),
            size_hint=size_hint,
            pos_hint=pos_hint,
            **kwargs,
```

```
        )

class PurpleButton(AppButton):
    '''Inheritance from parent class AppButton'''
    def __init__(
        self,
        text: str,
        font_size: int,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        super().__init__(
            text=text,
            font_size=font_size,
             #setting RGBA values
            background_color=RGBColour(90, 34, 139,1),
            size_hint=size_hint,
            pos_hint=pos_hint,
            **kwargs,
        )


#making a conection to the SQL database everytime app is run
conn = sqlite3.connect("users_db.sqlite")
c = conn.cursor()

c.execute(
    '''CREATE TABLE IF NOT EXISTS users (username TEXT PRIMARY
KEY, passwords TEXT)'''
)

class HasAccount(GridLayout, Screen):
    '''Checks if user has an account or if they need to make
one'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self.rows = 3
        self.spacing = 10
        self.padding = 80

        label = Label(text="Do you have an account?", color =
(0, 1, 0.52, 1))
        self.add_widget(label)

        #using GreenButton Class. - composition relation
        self.yes_button = GreenButton(
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            "YES",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.yes_button)
        self.yes_button.bind(on_press=self.on_yes_button)

        self.no_button = GreenButton(
            "NO",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.no_button)
        self.no_button.bind(on_press=self.on_no_button)


    def on_yes_button(self,instance):
        self.manager.current = 'login'


    def on_no_button(self,instance):
        self.manager.current = 'register'




class ErrorPage(GridLayout, Screen):
    '''Parent class from which all the error pages will inherit
and they will also override the on_back_button function'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self.rows = 2
        self.spacing = 10
        self.padding = 80

        self.label = Label(text="", color = (0, 1, 0.52, 1))
        self.add_widget(self.label)


        #Calling the GreenButton Class composition relation
        self.back_button = GreenButton(
            "***Try Again***",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        )
        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.on_back_button)

    def on_back_button(self,instance):
        pass


class LoginErrorPage(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "Invalid details, please try again."

    def on_back_button(self, instance):

        self.manager.current = 'login'


class RegisterErrorPage(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "Either the username is taken, or
details not filled properly.(Password > 5 chars)"

    def on_back_button(self, instance):

        self.manager.current = 'register'


class PlayerSearchError(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "----Error: not a valid input----"

    def on_back_button(self, instance):

        self.manager.current = 'search_bar'
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
class AccountRegisterLayout(GridLayout, Screen):
    # inheritance from Screen class (python's class)
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.cols = 2
        self.spacing = 20
        self.padding = 80
        self.username_button =  GreenButton(
                "Username",
                35,
                (0.3, 0.1),
                {"x": 0.2, "y": 0.05},
            )
        self.add_widget(self.username_button)
        self.username = TextInput(multiline=False,size_hint =
(0.5,0.002))
        self.add_widget(self.username)
        self.password_button = PurpleButton(
                "Password",
                35,
                (0.1, 0.1),
                {"x": 0.2, "y": 0.05}
            )
        self.add_widget(self.password_button)
        self.password = TextInput(multiline=False,size_hint =
(0.5,0.002),password=True)
        self.add_widget(self.password)
        self.register = GreenButton(
            "Register",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.005}
        )
        self.register.bind(on_press=self.register_user)
        self.add_widget(self.register)

        self.back_button = GreenButton(
            "Back",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )

        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.go_back)
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        self.fpl_logo = Image(source="fpl_logo.png", size_hint =
(0.5,0.5))
        self.add_widget(self.fpl_logo)

    def register_user(self, instance):
        '''Insert the user's information into the database'''

        conn = sqlite3.connect("users_db.sqlite")
        c = conn.cursor()

        username = self.username.text
        password = self.password.text
        print(password)
        self.valid_password = False
        self.valid_username = False
        #validation of whether username and password are valid
inouts or not
        if not username.strip():
            # can't be null inputs or empty spaces
            self.manager.current = 'register_error'
        else:
            self.valid_username = True

        if not password.strip():

            self.manager.current = "register_error"
        elif len(password) < 6:

            self.manager.current = "register_error"

        else:
            self.valid_password = True

        if self.valid_password == True and self.valid_username
== True:
            c.execute("SELECT * FROM users WHERE username=?",
(username,))
            user = c.fetchone()
            #checking if username already exists
            if user is not None:
                self.manager.current = 'register_error'
            else:
                #using hashing to do encryption to protect user
info
                hashed_password =
hashlib.sha256(password.encode())
                hashed_password = hashed_password.hexdigest()
                c.execute(
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                    "INSERT INTO users (username, passwords)
VALUES (?, ?)",
                    (username, hashed_password),
                )
                conn.commit()
                print("Record inserted successfully.")
                self.manager.current = "login"

    def go_back(self,instance):
        self.manager.current = 'has_account'


class LoginLayout(GridLayout, Screen):
    """A sub class of Parent python class called Gridlayout -
login in screen"""

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.cols = 2
        self.spacing = 20
        self.padding = 80
        self.add_widget(
            GreenButton(
                "Username",
                35,
                (0.3, 0.1),
                {"x": 0.2, "y": 0.05},
            )
        )
        self.username = TextInput(multiline=False,size_hint =
(0.5,0.002))
        self.add_widget(self.username)
        self.add_widget(

            PurpleButton(
                "Password",
                35,
                (0.1, 0.1),
                {"x": 0.2, "y": 0.05}
            )
        )
        self.password = TextInput(multiline=False,size_hint =
(0.5,0.002),password=True)


        self.add_widget(self.password)
        self.register = GreenButton(
            "Login",
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.005}
        )
        self.register.bind(on_press=self.login)
        self.add_widget(self.register)

        self.back_button = GreenButton(
            "Back",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )

        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.go_back)
        self.fpl_logo = Image(source="fpl_logo.png", size_hint =
(0.5,0.5), pos_hint = (1,0.5))
        self.add_widget(self.fpl_logo)

    def login(self, instance):
        # Check if the username and password match a user in the
database
        conn = sqlite3.connect("users_db.sqlite")
        c = conn.cursor()

        print("Login Input Hashed")
        self.password_to_hash = self.password.text
        hashed = hashlib.sha256(self.password_to_hash.encode())
        self.hashed_password = hashed.hexdigest()
        print(self.hashed_password)
        c.execute(
            "SELECT * FROM users WHERE username=? AND
passwords=?",
            (self.username.text, self.hashed_password),
        )
        user = c.fetchone()

        if user:
            print("Login successful")

            self.manager.current = "menu"

            # ...

        else:
            self.manager.current = 'login_error'
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    def go_back(self,instance):
        self.manager.current = 'has_account'


class MainMenu(GridLayout, Screen):
    """Allows users to choose between options in main menu"""

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.cols = 1
        self.spacing = 10
        self.padding = 30
        self.fpl_logo = Image(source="fpl_logo.png")
        self.add_widget(self.fpl_logo)
        self.search_button = GreenButton(
            "Search",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.search_button)
        self.search_button.bind(on_press=self.open_search_screen
)

        self.players_info_button = GreenButton(
            "Players Information",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2},
        )

        self.add_widget(self.players_info_button)
        self.players_info_button.bind(on_press=self.player_info)

        self.my_team_button = GreenButton(
            "My Team",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )

        self.add_widget(self.my_team_button)
        self.my_team_button.bind(on_press=self.display_team)

        self.back_button = GreenButton(
            "Back",
            35,
            (0.1, 0.1),
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            {"x": 0.2, "y": 0.2}
        )

        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.go_back_login)

    def display_team(self, instance):
        self.manager.current = "433"

    def open_search_screen(self, instance):
        self.manager.get_screen('search_bar').previous =
self.name
        self.manager.current = "search_bar"

    def player_info(self, instance):
        self.manager.current = "player_info"

    def go_back_login(self, instance):
        self.manager.current = "login"


class SearchBar(GridLayout, Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.cols = 1

        self.spacing = 30
        self.padding = 80
        self.previous = ''
        self.prediction_num = 0
        # Create a text input widget
        self.search_input = TextInput(
            hint_text="Enter player name",
            size_hint = (0.1,0.1),

            multiline=False
        )
        self.fpl_logo = Image(source="fpl_logo.png")
        self.add_widget(self.fpl_logo)
        # Create a search button widget
        self.search_button = GreenButton(
            "Search",
            35,
            (0.1, 0.1),
            {"x": 0.5, "y": 0.5},
        )

        # Add the text input and search button to the layout
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        self.add_widget(self.search_input)
        self.search_button.bind(on_press=self.predict)
        self.add_widget(self.search_button)
        self.back_button = PurpleButton(
            "Back",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )

        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.go_back)

    def go_back(self, instance):

        self.manager.current = self.previous

    def search(self, instance, player_name):
        try:
                # Connect to the SQL database
            conn = sqlite3.connect("fpl_players_db.sqlite")
            c = conn.cursor()

                # Search for the player in the database
            c.execute(
                "SELECT * FROM fpl_player_final_table WHERE
web_name=?", (player_name,))

            result = c.fetchall()[-1]

                # return result

            if result:

                return result

        except Exception as e:
            self.manager.current = 'search_error'

        finally:
            conn.close()

    def _get_player_average_and_sum(self, player_name):
       #cross-table parameterised SQL and aggregate SQL
functions taking place
        conn = sqlite3.connect("fpl_players_db.sqlite")
        cur = conn.cursor()
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        #getting average points this season by specific player
depending on the search
        query = """
        SELECT AVG(total_points)
        FROM fpl_player_final_table
        WHERE web_name = ?
        GROUP BY web_name
        """


        average = cur.execute(
            query, (player_name,)
        ).fetchall()

        print(type(average))



        conn.close()
        conn = sqlite3.connect("fpl_players_db.sqlite")
        cur = conn.cursor()
        #getting total points this season for specific player
depending on the search.
        query = """
        SELECT SUM(fpl_player_stats_table.total_points)
        FROM fpl_player_table
        INNER JOIN fpl_player_stats_table
        ON fpl_player_table.element =
fpl_player_stats_table.element
        WHERE fpl_player_table.web_name = ? AND
fpl_player_stats_table.web_name = ?
        """
        season_points = cur.execute(
            query, (player_name, player_name)
        ).fetchall()  # fetchall() retrieves all the rows
returned from SELECT statement


        return average, season_points


    def predict(self, instance):
        player_name = self.search_input.text
        arr = self.search(instance, player_name)
        if arr == None:
            self.manager.current = 'search_error'

        else:
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        # Load the saved model
            loaded_model =
tensorflow.keras.models.load_model("fpl_predictor.h5")

            # Get the new sample data
            new_sample = np.array([[arr[:6]]])
            new_sample = new_sample.reshape(-1, 6)

            # Scale the new sample data using the same scaler
that was used to scale the training data
            scaler = StandardScaler()

            data = np.loadtxt("training.csv", delimiter=",")

            # Load data from CSV file

            # Split data into input and output
            X = data[:, :6]
            y = data[:, 6]

            # Split data into training and testing sets
            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2)

            # Scale the input data
            scaler = StandardScaler()
            X_train = scaler.fit_transform(X_train)
            X_test = scaler.transform(X_test)

            # scaler.fit(data)
            new_sample = scaler.transform(new_sample)

            # Make predictions using the loaded model

            prediction = loaded_model.predict(new_sample)
            int_array = prediction.astype(int)
            prediction = int_array[0]
            self.prediction_num = prediction.item()
            prediction_val = self.prediction_num
            print(prediction_val)
            display_screen =
self.manager.get_screen('display_prediction')
            average, season_points =
self._get_player_average_and_sum(player_name)
            display_screen.update_score_label(player_name,
prediction_val, average, season_points)
            self.manager.current = 'display_prediction'
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
class DisplayPrediction(GridLayout,Screen):

    def __init__(self, **kwargs):

        super().__init__(**kwargs)

        self.rows = 2
        self.spacing = 10
        self.padding = 80
        self.previous = ''

        # print(prediction)

        self.label = Label(text=f"No Prediction Found!!", color
= (0, 1, 0.52, 1))
        self.add_widget(self.label)


        self.back_button = GreenButton(
            "Back",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.on_back_button)

    def update_score_label(self, player_name, score, average,
season_points):
        self.label.text = f"===== {player_name} Point
Prediction: {score} points=====\n=====
points this season: {average}. \n {player_name} total points
this season: {season_points}"

    def on_back_button(self,instance):
        self.manager.current = 'search_bar'

class UserTeamLayout(GridLayout, Screen):
    def __init__(self, **kwargs):

        super().__init__(**kwargs)
        self.cols = 3
        self.spacing = 10
        self.padding = 30
        # Add buttons to the layout

        self.gkbutton = GreenButton(
                "GK",
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.gkbutton)
        self.gkbutton.bind(on_press=self.open_search_screen)
        self.cb1Button = GreenButton(
            "CB",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.cb1Button)
        self.cb1Button.bind(on_press=self.open_search_screen)
        self.cb2Button = GreenButton(
            "CB",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.cb2Button)
        self.cb2Button.bind(on_press=self.open_search_screen)
        self.lbButton = GreenButton(
            "LB",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.lbButton)
        self.lbButton.bind(on_press=self.open_search_screen)

        self.rbButton = GreenButton(
            "RB",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.rbButton)
        self.rbButton.bind(on_press=self.open_search_screen)
        self.cdmButton = GreenButton(
            "CDM",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.cdmButton)
        self.cdmButton.bind(on_press=self.open_search_screen)
        self.cmButton = GreenButton(
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            "CM",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.cmButton)
        self.cmButton.bind(on_press=self.open_search_screen)

        self.camButton = GreenButton(
            "CAM",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.camButton)
        self.camButton.bind(on_press=self.open_search_screen)

        self.lwButton = GreenButton(
            "LW",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.lwButton)
        self.lwButton.bind(on_press=self.open_search_screen)

        self.rwButton = GreenButton(
            "RW",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.rwButton)
        self.rwButton.bind(on_press=self.open_search_screen)

        self.stButton = GreenButton(
            "ST",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.stButton)
        self.stButton.bind(on_press=self.open_search_screen)

        self.back_button = PurpleButton(
            "Back",
            35,
            (0.1, 0.1),
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.go_back_menu)

    def open_search_screen(self, instance):
        self.manager.get_screen('search_bar').previous =
self.name
        self.manager.current = "search_bar"
        self.first_instance = LoginLayout()



    def go_back_menu(self, instance):

        self.manager.current = 'menu'


# Connect to the database
conn = sqlite3.connect("fpl_players_db.sqlite")
cursor = conn.cursor()

# Fetch data from the SQL table
cursor.execute("SELECT * from fpl_player_table")
players = cursor.fetchall()


class PlayerInfoLayout(GridLayout, Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Connect to the SQLite database and retrieve the data
        conn = sqlite3.connect("fpl_players_db.sqlite")
        cur = conn.cursor()
        cur.execute("SELECT * FROM sorted_table")
        data = cur.fetchall()

        # Set the number of columns in the table
        self.cols = len(data[0])
        # Calculate the minimum height of the label layout
        self.minimum_height = len(data) * 100
        # Create a layout to hold the labels
        self.back_button = GreenButton(
            "Back",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        self.back_button.bind(on_press=self.go_back_to_menu)
        self.add_widget(self.back_button)

        scroll_view = ScrollView(size_hint=(1, 0.9))
        scroll_content = GridLayout(size_hint_y=None)
        scroll_content.bind(minimum_height=scroll_content.setter
('height'))

        label_layout = GridLayout(
            cols=self.cols, size_hint_y=None,
height=self.minimum_height
        )
        label_layout.bind(minimum_height=label_layout.setter("he
ight")
        # Loop through the data and add a label for each row
        for row in data:
            for value in row:
                label = Label(
                    text=str(value), color=(0, 1, 0.52, 1),
size_hint_y=None, height=100
                )
                label_layout.add_widget(label)

        # Add the label layout to the ScrollView

        scroll_view.add_widget(label_layout)
        self.add_widget(scroll_view)

    def go_back_to_menu(self,instance):
        self.manager.current = 'menu'

class PremierLeagueApp(App):  # inherits from parent class
App(kivy's class)
    """Main class, through which screens are defined so they can
be switched between"""


  def build(self):

        '''Uses Screen Manager to name each class as a screen so
I can switch between them'''
        screen_manager = ScreenManager()
        screen_manager.add_widget(HasAccount(name="has_account")
)
        screen_manager.add_widget(AccountRegisterLayout(name="re
gister"))
        screen_manager.add_widget(LoginLayout(name="login"))
        screen_manager.add_widget(MainMenu(name="menu"))
```

```
        screen_manager.add_widget(SearchBar(name="search_bar"))
    screen_manager.add_widget(PlayerInfoLayout(name="player_info
"))      screen_manager.add_widget(LoginErrorPage(name="login_err
or"))        screen_manager.add_widget(RegisterErrorPage(name="re
gister_error"))         screen_manager.add_widget(PlayerSearchErr
or(name="search_error"))
        screen_manager.add_widget(UserTeamLayout(name="433"))
        screen_manager.add_widget(DisplayPrediction(name="displa
y_prediction"))
        screen_manager.current = "has_account"
        return screen_manager


if __name__ == "__main__":
    PremierLeagueApp().run()
```

Techniques I have used in above script with specific parts of code:

Cross-table parameterised SQL
Aggregate SQL functions
User/CASE-generated DDL script

```
    def search(self, instance, player_name):
        try:
                # Connect to the SQL database
            conn = sqlite3.connect("fpl_players_db.sqlite")
            c = conn.cursor()

                # Search for the player in the database
            c.execute(
                "SELECT * FROM fpl_player_final_table WHERE
web_name=?", (player_name,))

            result = c.fetchall()[-1]

                # return result

            if result:
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
                return result

        except Exception as e:
            self.manager.current = 'search_error'

        finally:
            conn.close()

    def _get_player_average_and_sum(self, player_name):
       #cross-table parameterised SQL and aggregate SQL
functions taking place
        conn = sqlite3.connect("fpl_players_db.sqlite")
        cur = conn.cursor()
        #getting average points this season by specific player
depending on the search
        query = """
        SELECT AVG(total_points)
        FROM fpl_player_final_table
        WHERE web_name = ?
        GROUP BY web_name
        """


        average = cur.execute(
            query, (player_name,)
        ).fetchall()

        print(type(average))



        conn.close()
        conn = sqlite3.connect("fpl_players_db.sqlite")
        cur = conn.cursor()
        #getting total points this season for specific player
depending on the search.
        query = """
        SELECT SUM(fpl_player_stats_table.total_points)
        FROM fpl_player_table
        INNER JOIN fpl_player_stats_table
        ON fpl_player_table.element =
fpl_player_stats_table.element
        WHERE fpl_player_table.web_name = ? AND
fpl_player_stats_table.web_name = ?
        """
        season_points = cur.execute(
            query, (player_name, player_name)
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        ).fetchall()  # fetchall() retrieves all the rows
returned from SELECT statement


        return average, season_points
```

Dynamic generation of objects based complex user-defined use of OOP model

```
class RGBColour:
    def __init__(self, r: int, g: int, b: int, a: float = 1):
        self.r: int = r
        self.g: int = g
        self.b: int = b
        self.alpha: float = a


class AppButton(Button):
    '''making a parent class from which I can have child buttons
easily'''
    def __init__(
        self,
        text: str,
        font_size: int,
        background_color: RGBColour,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        #kivy has rgba values which are the actual rgb values /
255
        background_color: Tuple = (
            background_color.r / 255,
            background_color.g / 255,
            background_color.b / 255,
            background_color.alpha,
        )
        super().__init__(
            #all of below will be attributes I will vary in
buttons.
            text=text,
            font_size=font_size,
            background_color=background_color,
            size_hint=size_hint,
            pos_hint=pos_hint,
```

```
            **kwargs,
        )

class GreenButton(AppButton):
    '''Inheritance from parent class AppButton'''
    def __init__(
        self,
        text: str,
        font_size: int,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        super().__init__(
            text=text,
            font_size=font_size,
            #setting RGBA values
            background_color=RGBColour(46, 204, 113,1),
            size_hint=size_hint,
            pos_hint=pos_hint,
            **kwargs,
        )

class PurpleButton(AppButton):
    '''Inheritance from parent class AppButton'''
    def __init__(
        self,
        text: str,
        font_size: int,
        size_hint: Tuple,
        pos_hint: Dict,
        **kwargs,
    ):
        super().__init__(
            text=text,
            font_size=font_size,
             #setting RGBA values
            background_color=RGBColour(90, 34, 139,1),
            size_hint=size_hint,
            pos_hint=pos_hint,
            **kwargs,
        )
```

Above and below are just two examples of technique as the whole script does this technique of OOP.

```
class ErrorPage(GridLayout, Screen):
    '''Parent class from which all the error pages will inherit
and they will also override the on_back_button function'''
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self.rows = 2
        self.spacing = 10
        self.padding = 80

        self.label = Label(text="", color = (0, 1, 0.52, 1))
        self.add_widget(self.label)


        #Calling the GreenButton Class composition relation
        self.back_button = GreenButton(
            "***Try Again***",
            35,
            (0.1, 0.1),
            {"x": 0.2, "y": 0.2}
        )
        self.add_widget(self.back_button)
        self.back_button.bind(on_press=self.on_back_button)

    def on_back_button(self,instance):
        pass


class LoginErrorPage(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "Invalid details, please try again."

    def on_back_button(self, instance):

        self.manager.current = 'login'


class RegisterErrorPage(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "Either the username is taken, or
details not filled properly.(Password > 5 chars)"

    def on_back_button(self, instance):
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
        self.manager.current = 'register'


class PlayerSearchError(ErrorPage):
    '''Inhertiance from Error Page parent class'''
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Update label text
        self.label.text = "----Error: not a valid input----"

    def on_back_button(self, instance):

        self.manager.current = 'search_bar'
```

Hashing
User/CASE-generated DDL script

```
def register_user(self, instance):
        '''Insert the user's information into the database'''

        conn = sqlite3.connect("users_db.sqlite")
        c = conn.cursor()

        username = self.username.text
        password = self.password.text
        print(password)
        self.valid_password = False
        self.valid_username = False
        #validation of whether username and password are valid
inouts or not
        if not username.strip():
            # can't be null inputs or empty spaces
            self.manager.current = 'register_error'
        else:
            self.valid_username = True

        if not password.strip():

            self.manager.current = "register_error"
        elif len(password) < 6:
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            self.manager.current = "register_error"

        else:
            self.valid_password = True

        if self.valid_password == True and self.valid_username
== True:
            c.execute("SELECT * FROM users WHERE username=?",
(username,))
            user = c.fetchone()
            #checking if username already exists
            if user is not None:
                self.manager.current = 'register_error'
            else:
                #using hashing to do encryption to protect user
info
                hashed_password =
hashlib.sha256(password.encode())
                hashed_password = hashed_password.hexdigest()
                c.execute(
                    "INSERT INTO users (username, passwords)
VALUES (?, ?)",
                    (username, hashed_password),
                )
                conn.commit()
                print("Record inserted successfully.")
                self.manager.current = "login"

    def go_back(self,instance):
        self.manager.current = 'has_account'




def login(self, instance):
        # Check if the username and password match a user in the
database
        conn = sqlite3.connect("users_db.sqlite")
        c = conn.cursor()

        print("Login Input Hashed")
        self.password_to_hash = self.password.text
        hashed = hashlib.sha256(self.password_to_hash.encode())
        self.hashed_password = hashed.hexdigest()
        print(self.hashed_password)
        c.execute(
```

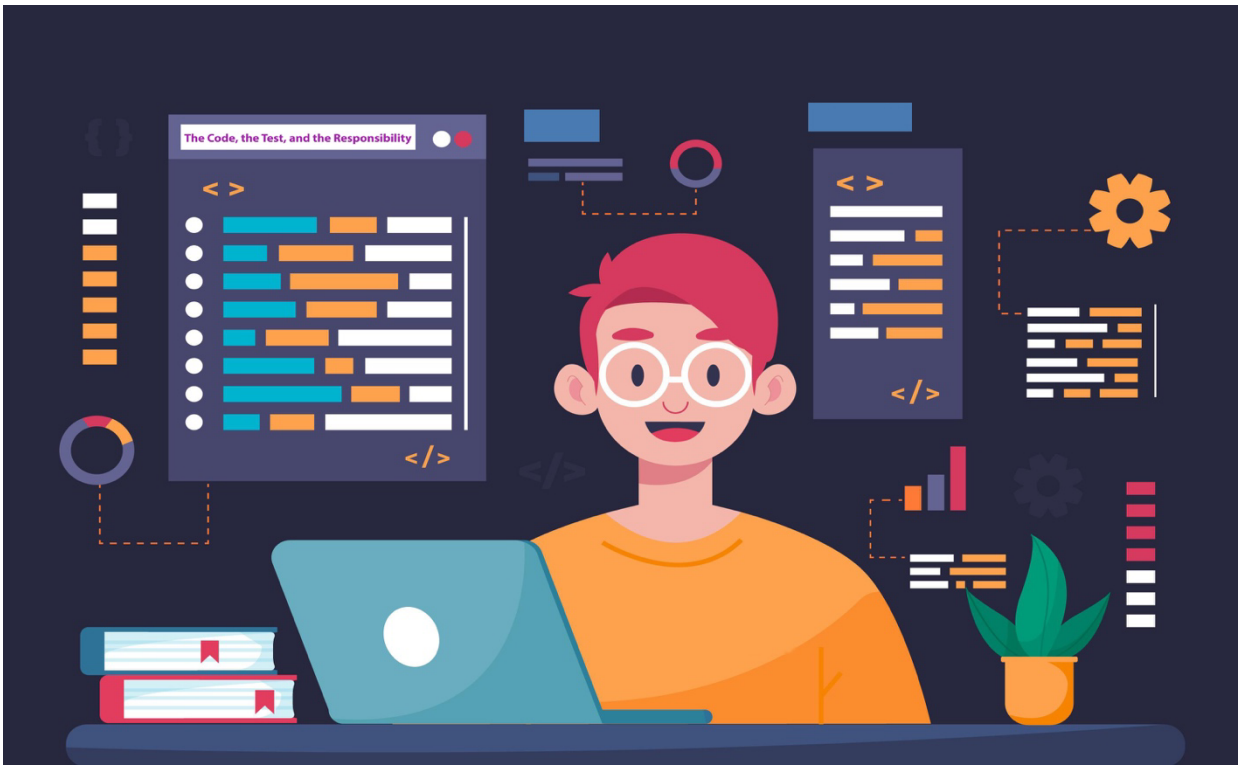

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

```
            "SELECT * FROM users WHERE username=? AND
passwords=?",
            (self.username.text, self.hashed_password),
        )
        user = c.fetchone()

        if user:
            print("Login successful")

            self.manager.current = "menu"

            # ...

        else:
            self.manager.current = 'login_error'

    def go_back(self,instance):
        self.manager.current = 'has_account'
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Testing

Prannvat Singh



Testing Overview

My NEA required me to do four main things:

1. Collect Data
2. Make Machine Learning Model
3. Make GUI
4. Combine the above and make a functional product

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Iterative Testing

Throughout the process of coding my project, I consistently kept testing my code. This meant that a lot of my back end was always being checked, and now at this stage in my testing, all my collection of data and my model accuracy works efficiently and correctly.

DATA_T_ITER

For OBJECTIVE 1 {Analysis}, I needed to gather the data, and I needed to test if I was getting the right data.

To get the set of data I would train my model with I took requests from the Premier League API and using JSON parsing, I converted them into Panda's data frames, which I could then convert to SQL tables.

```
~/Documents/Projects/fpl-points-predictor main* 3m 55s
fpl-venv ❯ /Users/prannvatsingh/Documents/Projects/fpl-points-predictor/fpl-venv/bin/python
100%|
     element    web_name singular_name_short  total_points  goals_scored  \
317      318     Haaland                 FWD           171            25
426      427        Kane                 FWD           148            17
356      357    Trippier                 DEF           143             1
334      335    Rashford                 MID           128            11
6          7    Ødegaard                 MID           119             8
..       ...         ...                 ...           ...           ...
562      563    Chambers                 DEF             0             0
739      740       Hegyi                 GKP             0             0
726      727     Souttar                 DEF            -1             0
152      153     Tomkins                 DEF            -1             0
689      690  João Félix                 FWD            -2             0

     assists  goals_conceded
317        3              20
426        5              30
356        5              10
334        4              25
6          6              16
..       ...             ...
562        0               0
739        0               0
726        0               2
152        0               5
689        0               2

[740 rows x 7 columns]
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

As part of OBJECTIVE 2 {Analysis}, I then added all the collected data to SQL tables. This was easy to iteratively test as I could use the SQLite explorer extension on VSCode, which just showed me the full tables that I made. Below are examples of tables as proof.

SQLite

SQL ▾ 1 / 273 1 – 50 of 13632

id_player	opponent_team	was_home	round	transfers_in	selected	total_points	element	web_name	singular_name_short
3	7	0	1	0	48303	2	3	Xhaka	MID
3	10	1	2	9001	65418	12	3	Xhaka	MID
3	3	0	3	137326	216726	6	3	Xhaka	MID
3	9	1	4	77459	267951	2	3	Xhaka	MID
3	2	1	5	49435	288460	2	3	Xhaka	MID
3	14	0	6	50721	308627	2	3	Xhaka	MID
3	4	0	8	11300	302039	5	3	Xhaka	MID
3	18	1	9	32640	317511	9	3	Xhaka	MID
3	12	1	10	83754	390185	2	3	Xhaka	MID
3	11	0	11	26252	381694	3	3	Xhaka	MID
3	17	0	13	34061	368620	10	3	Xhaka	MID
3	16	1	14	77987	438068	3	3	Xhaka	MID
3	6	0	15	28099	429403	3	3	Xhaka	MID
3	20	0	16	23835	420636	1	3	Xhaka	MID
3	19	1	17	88192	396383	5	3	Xhaka	MID
3	5	0	18	15387	396368	2	3	Xhaka	MID
3	15	1	19	6461	371502	2	3	Xhaka	MID
3	18	0	20	9460	349243	3	3	Xhaka	MID
3	14	1	21	4547	330216	5	3	Xhaka	MID

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

As part of OBJECTIVE 3 as well as OBJECTIVE 2 {Analysis}, I needed to store user information on an SQL table. For my account system I needed to hash the user password as a form of encryption on the passwords so if there was a breach in the database, people's passwords would remain safe. I used sha256 for this encryption:

SQL ▼ 1 / 1 1 - 10 of 10

username	passwords
Prannvat	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c9
testing	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802
testing1	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802
leoisjustbetter	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802
prannvat	41dc043f7317996e8211f5ae9a00cf6dfd0154d106942ff8526316ff24c83b2
SMELL_E_PACK_E	170e54eb0358973bcc826819e9358438277f80e86aed8cd61a6f7bdca4de4df
BigBananaMuncher	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c9
BigBananaMuncher2	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c9
Usertesting101	e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221
Testing123	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c9

SQL ▾ 1 / 15

element	web_name	singular_name_short
1	Cédric	DEF
2	Leno	GKP
3	Xhaka	MID
4	Elneny	MID
5	Holding	DEF
6	Partey	MID
7	Ødegaard	MID
8	Tierney	DEF
9	Pépé	MID
10	White	DEF
11	Nketiah	FWD
12	Smith Rowe	MID
13	Saka	MID
14	Tomiyasu	DEF
15	Ramsdale	GKP
16	Gabriel	DEF
17	Tavares	DEF
18	Sambi	MID
19	Martinelli	MID
20	Pablo Marí	DEF

SQL ▾ 1 / 163

element	web_name	total_points
318	Haaland	170
427	Kane	145
357	Trippier	143
335	Rashford	120
7	Ødegaard	119
369	Almirón	114
13	Saka	111
80	Toney	111
301	De Bruyne	107
283	Salah	105
376	Pope	105
19	Martinelli	103
366	Schär	99
81	Raya	98
333	Fernandes	97
26	Saliba	93
104	Groß	93
107	March	93
225	Rodrigo	92

I used aggregate SQL functions as well as using cross-table parameterised so I could convert these tables and create a bigger table with more information.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Below is the table I used for retrieving the data from which predictions are made. Each row shows the points of the previous week and the data I need for the week:

SQL ▾ ❮ 1 / 273 ❯ 1 - 50 of 13632

id_player	opponent_team	was_home	round	transfers_in	selected	total_points	element	web_name	singular_name_short
3	7	0	1	0	48303	2	3	Xhaka	MID
3	10	1	2	9001	65418	12	3	Xhaka	MID
3	3	0	3	137326	216726	6	3	Xhaka	MID
3	9	1	4	77459	267951	2	3	Xhaka	MID
3	2	1	5	49435	288460	2	3	Xhaka	MID
3	14	0	6	50721	308627	2	3	Xhaka	MID
3	4	0	8	11300	302039	5	3	Xhaka	MID
3	18	1	9	32640	317511	9	3	Xhaka	MID
3	12	1	10	83754	390185	2	3	Xhaka	MID
3	11	0	11	26252	381694	3	3	Xhaka	MID
3	17	0	13	34061	368620	10	3	Xhaka	MID
3	16	1	14	77987	438068	3	3	Xhaka	MID
3	6	0	15	28099	429403	3	3	Xhaka	MID
3	20	0	16	23835	420636	1	3	Xhaka	MID
3	19	1	17	88192	396383	5	3	Xhaka	MID
3	5	0	18	15387	396368	2	3	Xhaka	MID
3	15	1	19	6461	371502	2	3	Xhaka	MID
3	18	0	20	9460	349243	3	3	Xhaka	MID
3	14	1	21	4547	330216	5	3	Xhaka	MID

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

MODEL_T_ITER

As part of OBJECTIVE 5 {Analysis}, I needed to create a neural network and continuously test it so I could get it to be more accurate.

I needed to change the batch number and epoch size of my model as well as the number of layers and nodes. I did this through trial and error and eventually got a relatively low and acceptable Test Loss value, I eventually chose an epoch size of 150, batch size of 34 and had three layers in my model. I initially attempted this with 2 layers and 100 epochs and a batch size of 30, this did not lead to a good test loss value. After several attempts and messing around with my model, I now believe that this is close to the best accuracy that my model will reach. Using TensorFlow and Keras was a great decision as it had a great interface, which allowed me to implement and the loss value it gives is very good meaning my predictions are reasonably accurate.

```
Epoch 120/150
321/321 [==============================] - 0s 277us/step - loss: 4.5271
Epoch 121/150
321/321 [==============================] - 0s 285us/step - loss: 4.5297
Epoch 122/150
321/321 [==============================] - 0s 281us/step - loss: 4.5271
Epoch 123/150
321/321 [==============================] - 0s 281us/step - loss: 4.5338
Epoch 124/150
321/321 [==============================] - 0s 280us/step - loss: 4.5226
Epoch 125/150
321/321 [==============================] - 0s 283us/step - loss: 4.5228
Epoch 126/150
321/321 [==============================] - 0s 285us/step - loss: 4.5192
Epoch 127/150
321/321 [==============================] - 0s 280us/step - loss: 4.5206
Epoch 128/150
321/321 [==============================] - 0s 279us/step - loss: 4.5264
Epoch 129/150
321/321 [==============================] - 0s 279us/step - loss: 4.5208
Epoch 130/150
321/321 [==============================] - 0s 279us/step - loss: 4.5145
Epoch 131/150
321/321 [==============================] - 0s 273us/step - loss: 4.5176
Epoch 132/150
321/321 [==============================] - 0s 275us/step - loss: 4.5169
Epoch 133/150
321/321 [==============================] - 0s 285us/step - loss: 4.5223
Epoch 134/150
321/321 [==============================] - 0s 283us/step - loss: 4.5125
Epoch 135/150
321/321 [==============================] - 0s 275us/step - loss: 4.5184
Epoch 136/150
321/321 [==============================] - 0s 280us/step - loss: 4.5046
Epoch 137/150
321/321 [==============================] - 0s 285us/step - loss: 4.5078
Epoch 138/150
321/321 [==============================] - 0s 282us/step - loss: 4.5079
Epoch 139/150
321/321 [==============================] - 0s 279us/step - loss: 4.5144
Epoch 140/150
321/321 [==============================] - 0s 280us/step - loss: 4.5073
Epoch 141/150
321/321 [==============================] - 0s 283us/step - loss: 4.5112
Epoch 142/150
321/321 [==============================] - 0s 281us/step - loss: 4.4953
Epoch 143/150
321/321 [==============================] - 0s 289us/step - loss: 4.5082
Epoch 144/150
321/321 [==============================] - 0s 282us/step - loss: 4.5076
Epoch 145/150
321/321 [==============================] - 0s 295us/step - loss: 4.5040
Epoch 146/150
321/321 [==============================] - 0s 276us/step - loss: 4.4965
Epoch 147/150
321/321 [==============================] - 0s 276us/step - loss: 4.5082
Epoch 148/150
321/321 [==============================] - 0s 285us/step - loss: 4.5013
Epoch 149/150
321/321 [==============================] - 0s 280us/step - loss: 4.4922
Epoch 150/150
321/321 [==============================] - 0s 280us/step - loss: 4.5006
Test loss: 4.174718379974365
```


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

ALL OF THESE TABLES REFERENCE MY TESTING VIDEO ON THE TIME STAMP. THIS VIDEO CAN BE FOUND ON YOUTUBE AT https://www.youtube.com/watch?v=ZBZTaUQHwCU&t=355s&ab_channel=MaidenErleghSchoolComputingDepartment

[In all the below tables, I have written inputs which may be different to my testing video, this is to show that this indeed works on not only the inputs I show in my video but all inputs.

I felt this was a good way of showing all possible attempts of crashing or infiltrating my app and how its robustness prevents that.]

GUI_T1

Asks user if they have an account or not, then lets them either make an account or login. Asks them the question:

"Do you have an account? "

Has two buttons YES and NO

TIME STAMP: 4:20

Test#	**Input**	**Expected Output**	**Actual Output**	**PASS/FAIL**	**COMMENT**
1	YES	Switches to Login Screen	Switches to Login Screen	PASS	No real way to fail this, as there are only two buttons you can possibly click

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

2	NO	Switches to Registration Screen	Switches to Registration Screen	PASS	

GUI_T2

Asking user to login:

"USERNAME, PASSWORD", (LOGIN BUTTON)

TIME STAMP: 4:27

Test#	**Input**	**Expected Output**	**Actual Output**	**PASS/ FAIL**	**COMMENT**
1	'Prannvat' '123456'	"Login Successful" Takes you to menu	"Login Successful" Takes you to menu	PASS	Already have a name with these details set, thus it allows me to enter
2	'' ''	Error Page	Error Page	PASS	Null inputs not accepted as they are not accepted as values to register with in the first place.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

3	'Prannvat' '1234'	Error Page	Error Page	PASS	Correct username but incorrect password, denies entry as expected.
4	'User000' 'Testing1234'	"Login Successful" Takes you to menu	"Login Successful" Takes you to menu	PASS	

GUI_T3

Asks user to register using a username and password

"USERNAME, PASSWORD", (REGISTER BUTTON)

TIME STAMP: 5:20

Test#	**Input**	**Expected Output**	**Actual Output**	**PASS/ FAIL**	**COMMENT**
1	'TestUser12345 Password test'	"Record inserted successfully" Opens menu page"	"Record inserted successfully" Opens menu page"	PASS	Password doesn't have a certain length limitation. (Not important as there is no sensitive information). All the passwords are hashed and encrypted.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

2	'Prannvat' '1234'	Error Page	Error Page	PASS	Already have an account with that username I have set up, as you can see in the video.
3	'' '123456'	Error Page	Error Page	PASS	The username is given no value or a bunch of spaces, which it doesn't allow.
4	'Usertest' '234'	Error Page	Error Page	PASS	The password is given less than 6 chars or no value or a bunch of spaces, which it doesn't allow.
5	'User000' 'Testing 1234'	"Record inserted successfully" Opens menu page"	"Record inserted successfully" Opens menu page"	PASS	

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

GUI_T4

Clicking different Buttons in the main menu.

TIME STAMP 4:44

Test#	Input	Expected Output	Actual Output	PASS/ FAIL	COMMENT
1	Press 'Back' button	Takes you back to login page	Takes you back to login page	PASS	All these pages allow you to come back to menu.
2	Press 'Search' button	Takes you to search page	Takes you to search page	PASS	
3	Press 'Players Info' button	Take you to page with all the players and their total points. Should show players in order of most points	Take you to page with all the players and their total points. Shows players in order of most points	PASS	Good little feature to see which players are doing best

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

4	Press 'My Team' button	Takes you to page with all positions in team which you can also search from	Takes you to page with all positions in team which you can also search from	PASS	Nice feature for user. In reality, no different to search screen.

GUI_T5

Searching for player and getting a point prediction as well as average and total points for that player

TIME STAMP: 5:50

Test#	**Input**	**Expected Output**	**Actual Output**	**PASS/ FAIL**	**COMMENT**
1	'Haaland'	Goes to prediction display page	Goes to prediction display page	PASS	
2	'Ødegaard'	Goes to prediction display page	Goes to prediction display page	PASS	Accents must be correct in name

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

	'Messi'	Error Page	Error Page	PASS	
4	''	Error Page	Error Page	PASS	

GUI_T6

Clicking Position Button in MyTeamLayout

TIME STAMP: 7:30

Test#	**Input**	**Expected Output**	**Actual Output**	**PASS/ FAIL**	**COMMENT**
1	GK	Takes you to search bar	Takes you to search bar	PASS	

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

2	CB	Takes you to search bar	Takes you to search bar	PASS	
	RW	Takes you to search bar	Takes you to search bar	PASS	
4	Back	Takes you to Main menu	Takes you to Main menu	PASS	

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Evaluation

Prannvat Singh

After a very eventful several months of hard work, my NEA technical solution is ready. The process of completing my project had a lot of ups and downs, a lot of mental struggles and frustrations and amazing highs when I fixed a problem through a lot of perseverance. I am proud of my project; however, it is important to truly evaluate what I have achieved and how it compares to what I initially set out to achieve.

Let's see how my actual product compares against my initial objectives {Analysis} and if I have achieved than them or failed to achieve them.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

What my client wanted?

The initial proposal was an easy-to navigate app, which gave them information about all the players on how many points they have this season, and also the ability to give a reasonable prediction on how many points any player in the premier league will get.

Achieved? ✅

These requests have indeed been completed in this NEA, not only does my app predict points and show how many points each player has in the league, but it this through a simplistic yet appealing GUI, which makes my product very user-friendly. The prediction is done from my machine learning model, which has learnt from over 15000 sets of data and this model will only get better with time. It is instant and happens in the backend while the user can just search a name and get their prediction nicely presented to them, in fact, with more information about that, like the player's average points in a game week this season.

OBJECTIVE 1

REQUESTING AND ACCESSING THE DATA FROM PREMIER LEAGUE API AND PARSING AS JSON

Without, this first objective, none of my project would be possible. Data handling is the backbone of my NEA. In order to make a neural network, which would predict how many points a player will get, I needed to collect a lot of data on each player. To do this I needed to utilise the Premier League API.

Achieved? ✅

As this API meant all my data would be a JSON object, once I had requested the data from the API and gathered the data through my network with this API; I parsed the data into many dictionaries, which I could then use, I gathered the required data, by which I would train the model. As the surveys suggested {Analysis} I used the players form through the usage of how many points they had been getting in each round. I also used the data for how many people were selecting the player and buying the players as

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

well as who their opponent was. I considered home ground to be a major factor too hence I used data on whether the player was playing in their home stadium or not. I then gathered all of this data from the past season and also how many points each player got in each week dependent on above factors. I then converted this into a .csv file which I used to train my model and also converted into an SQL table so I could then use cross-table parameterised SQL and aggregate SQL to gather average points for player and total points in the season. If I wish to update my product in the future, I can easily also make more GUI's which display very detailed information about all players, however as this wasn't what my client needed and what I set out to do, I didn't deem it necessary to do for my NEA.

OBJECTIVE 2

USING SQL TO STORE DATA COLLECTED FROM API, JOIN TABLES AND ACCESS PLAYER INFORMATION FROM

As I mentioned above, I converted my data into SQL tables. This was tricky as I had to create several Pandas' data frames from the API as there was a variety of data I collected from different URLs. This would allow me to easily manage the data and use it properly.

Achieved? ✅

I created 6 tables in total, one for the user login details, the rest on footballers' information. In order to create these, I decided to use SQLite to manage my databases, as that provides a portability, which MySQL Workbench does not. I used cross-table parameterised SQL in my tables as I needed to join together SQL tables to create different tables. This also allowed me to have 3NF Normalised tables which I could then. Join together to make a big table from which I extract my information from which I predict the points of a player. In addition to this I also display average and sum points of a player when I am showing the prediction through aggregate SQL. One of my GUI's displays information which, I got from the API in the form of a list. I merge sorted this in order of total points and added it onto an SQL table. I realise that I could have used an SQL command which could have ordered my table for me, however I decided to use a merge sort to display my understanding of one.

OBJECTIVE 3

ACCOUNT REGISTRATION- MAKE AN ENCRYPTED AND AUTHENTICATED LOGIN AND REGISTRATION SYSTEM

I wanted to add an account registration system on my app. I believe that this is an important base for further additions to my product as I can use this to store specific teams that users make in their specific accounts, and this is a good way to set that up. Though that isn't something my NEA contains, my account registration system is important as it makes future updates easy to implement.

Achieved? ✅

My NEA has a secure account registration system. Users can make an account and login with the username and password they made at registration. No two people are allowed the same username and the passwords are all encrypted on my database making the user information secure. In addition to this, I use parameterized SQl to prevent SQL injections from people with malicious intentions. This can be made even more impenetrable in the future by using sanitization of inputs, however my system is acceptably secure none the less. This is all incorporated into my GUI like everything else. All the username and encrypted passwords are stored on a 3NF SQL table.

OBJECTIVE 4

GRAPHICAL USER INTERFACE – MAKING THE BACKEND OF NEURAL NETWORK BE DISPLAYED ON AN APPEALING USER INTERFACE WHICH IS EASY TO USE FOR USER

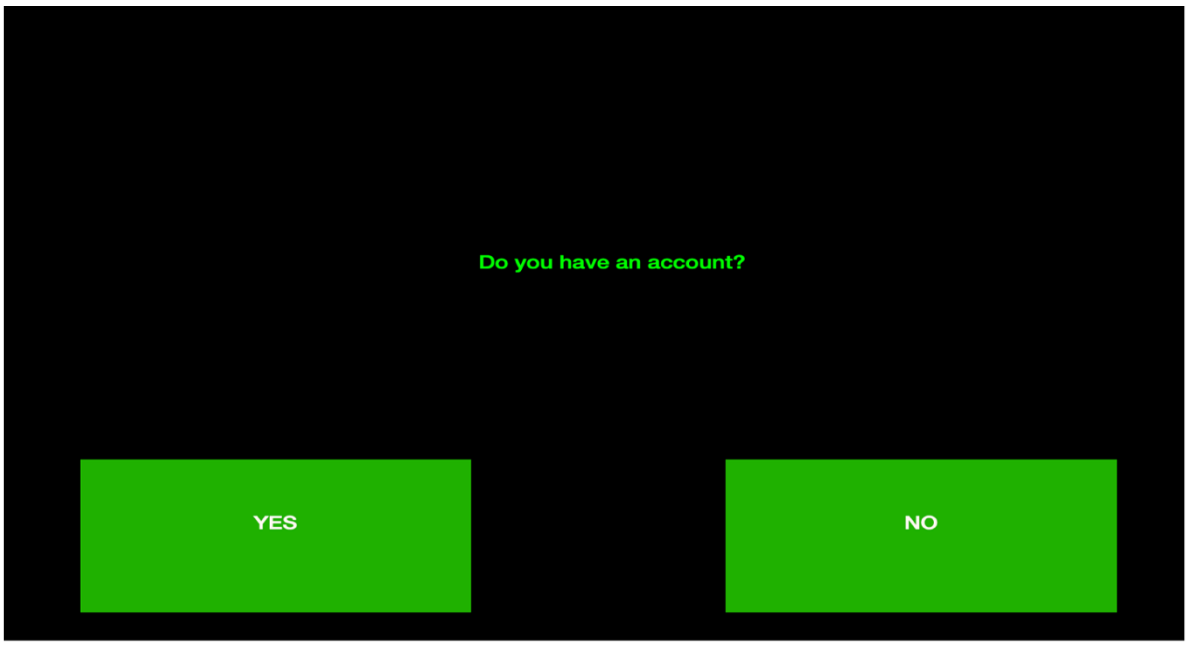
My GUI was a very crucial feature of my app, didn't need to be extensive; just a simple, appealing GUI, which enabled my end-user to easily use their desired main feature of predicting the FPL points of all premier league players. I decided to use Kivy as my GUI toolkit, as it enabled me to do Object-Oriented-Programming extensively and efficiently. My initial design was very clean and simple, yet appealing. However, once I began implementing my GUI, I realised that it was quite difficult to make a functioning GUI how I wanted to. This led to me producing a very poor GUI, which after some criticism from people I asked, I decided to scrap. I then persevered and eventually produced a GUI very similar to my design {Documented Design}. In fact, I have made additions to my initial design and added error pages GUI's (I did not design them in {Documented Design} but had a plan for them in terms of a UML diagram), as well as GUI which

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

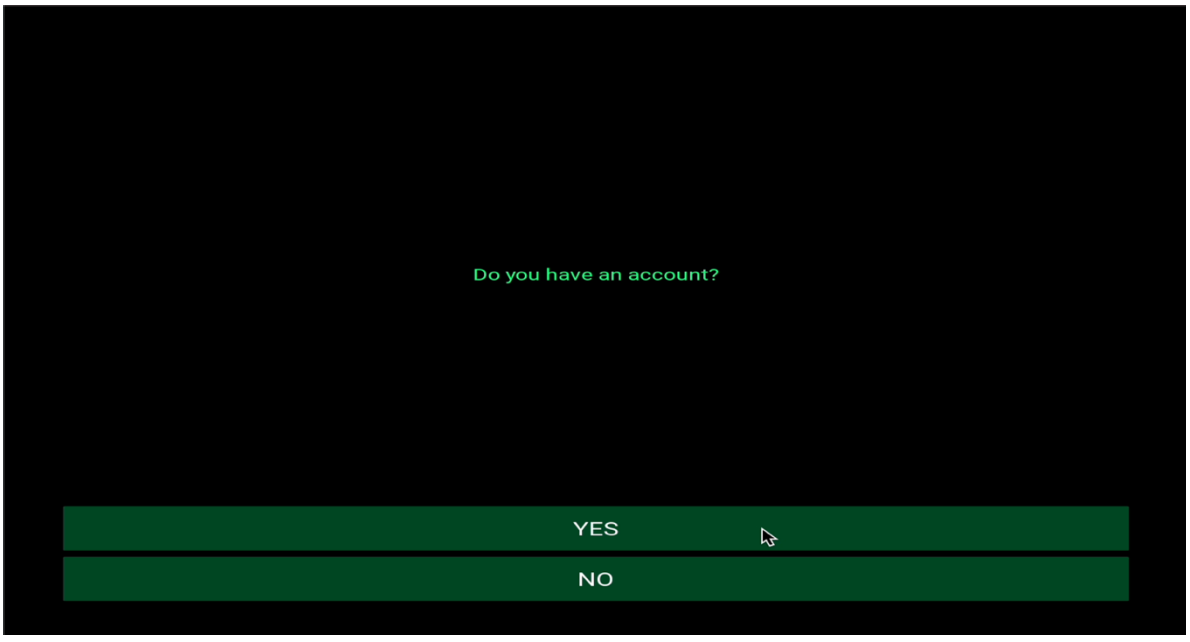
shows the prediction of a player and the player's average and total points.

On the next pages, you will be able to see the comparison between my GUI Design and my final GUI.

Design

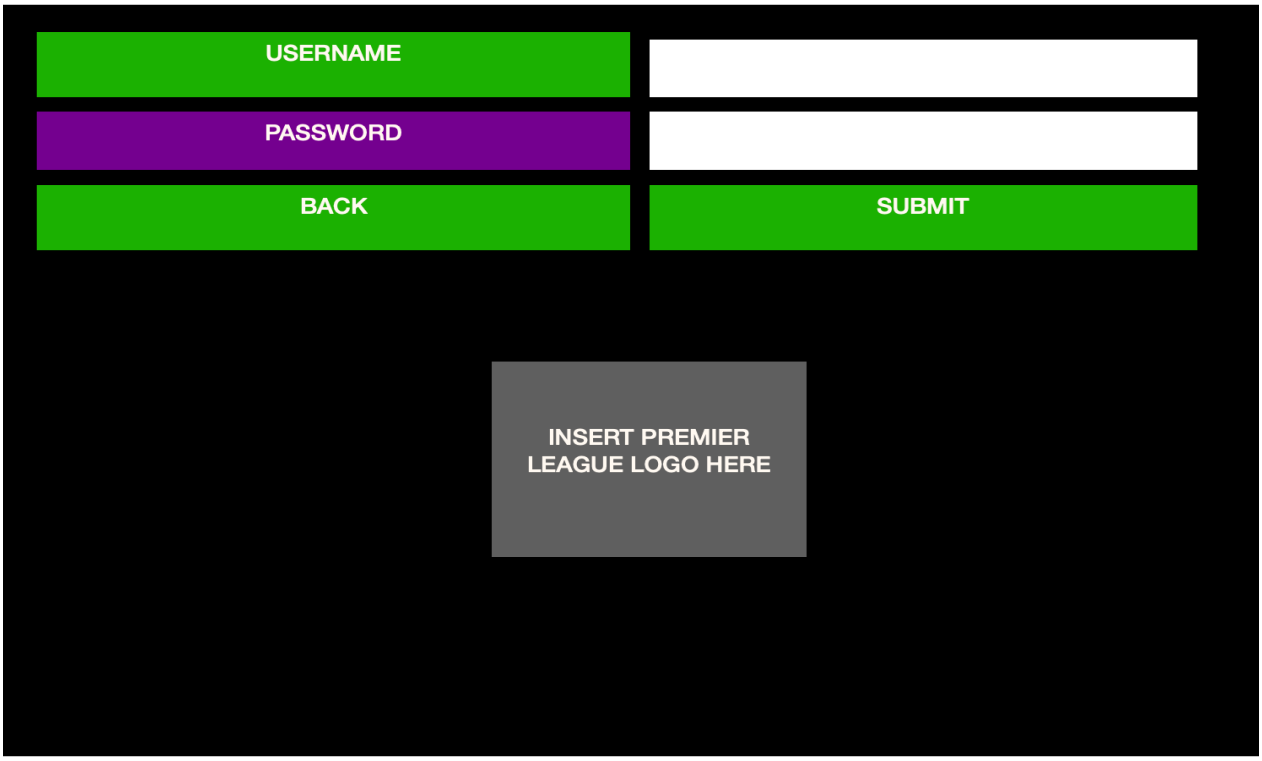


Final GUI

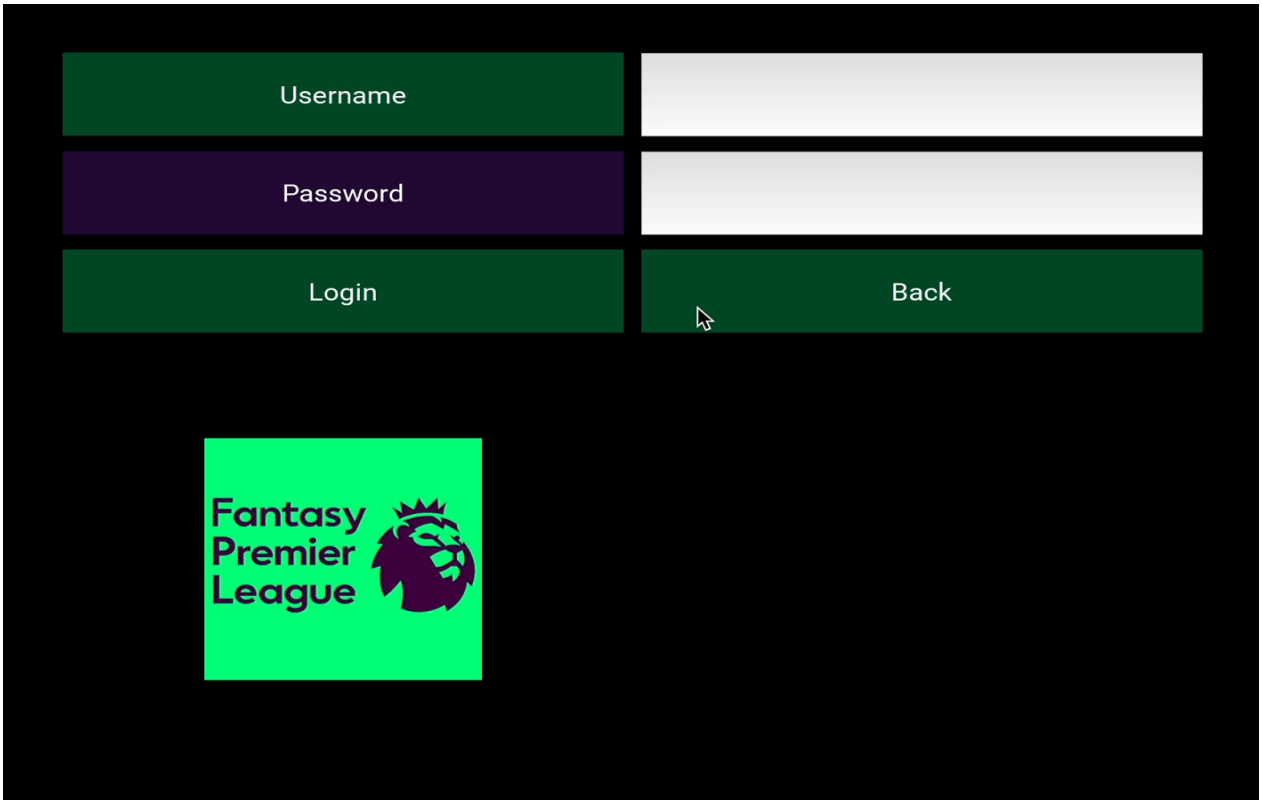


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Design

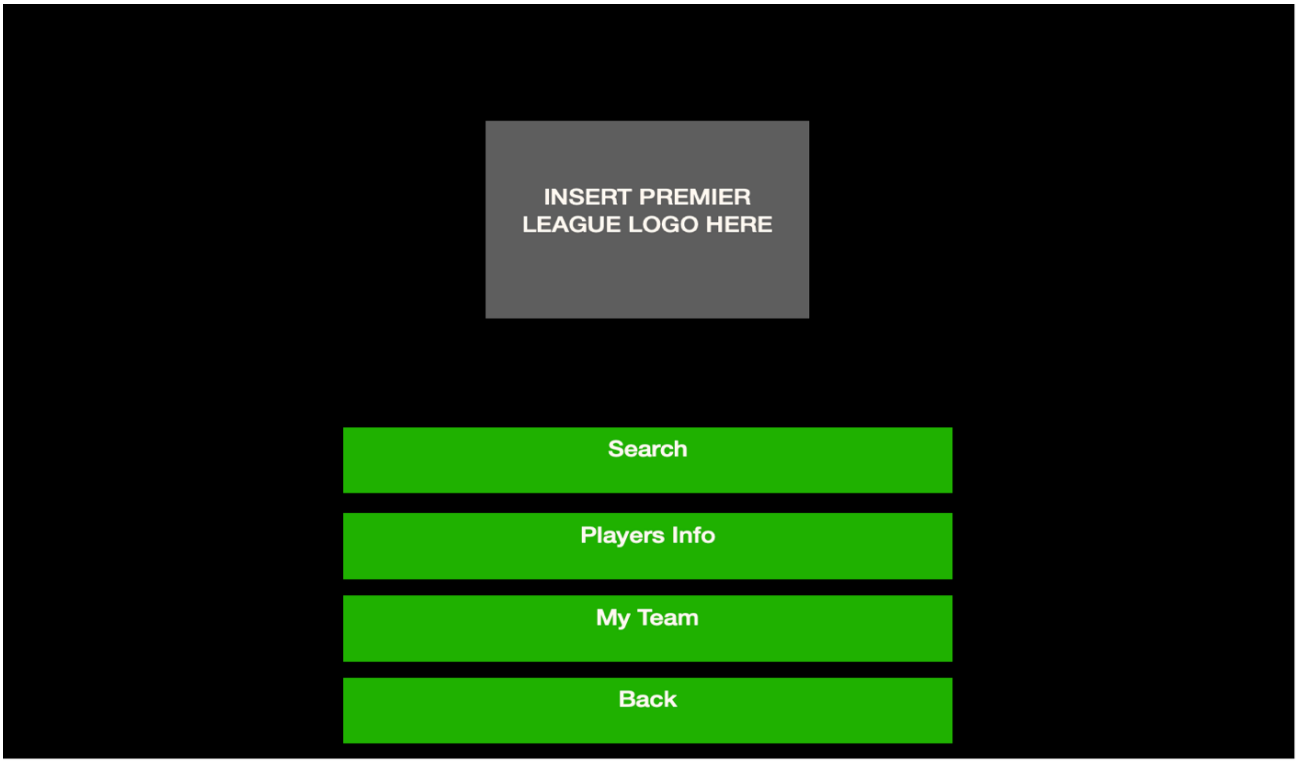


Final GUI

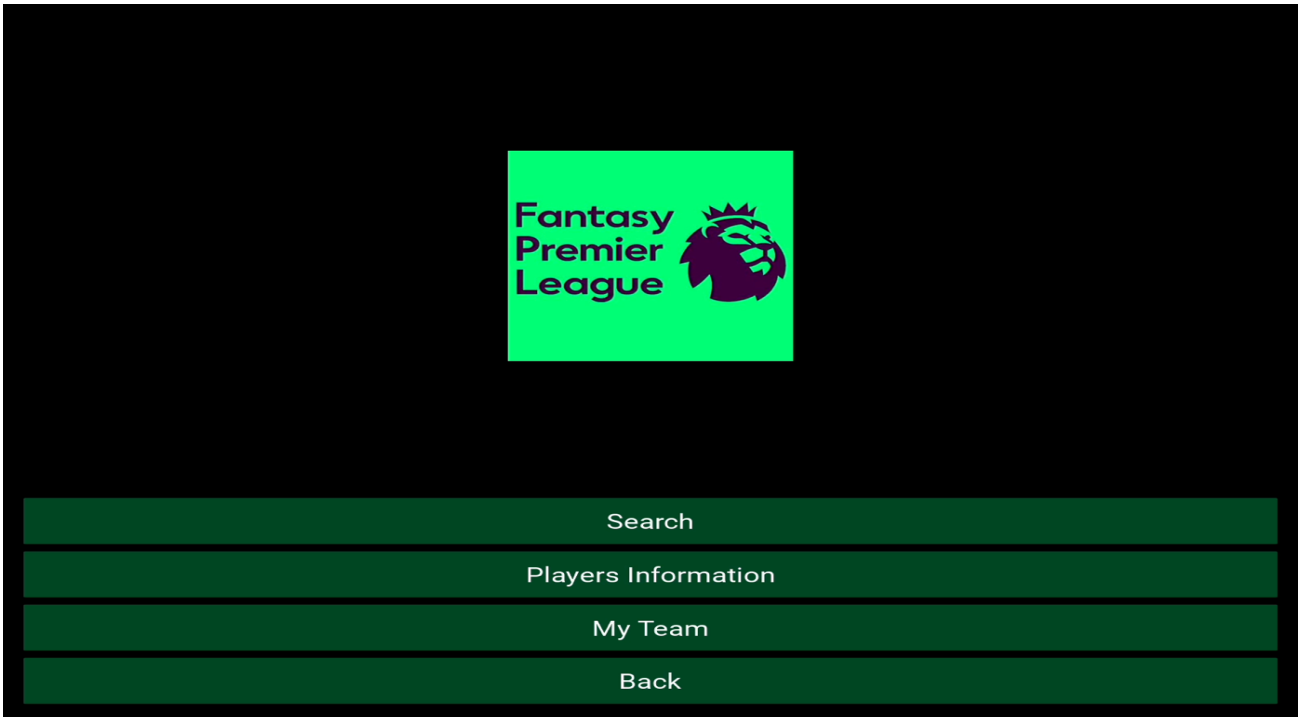


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Design

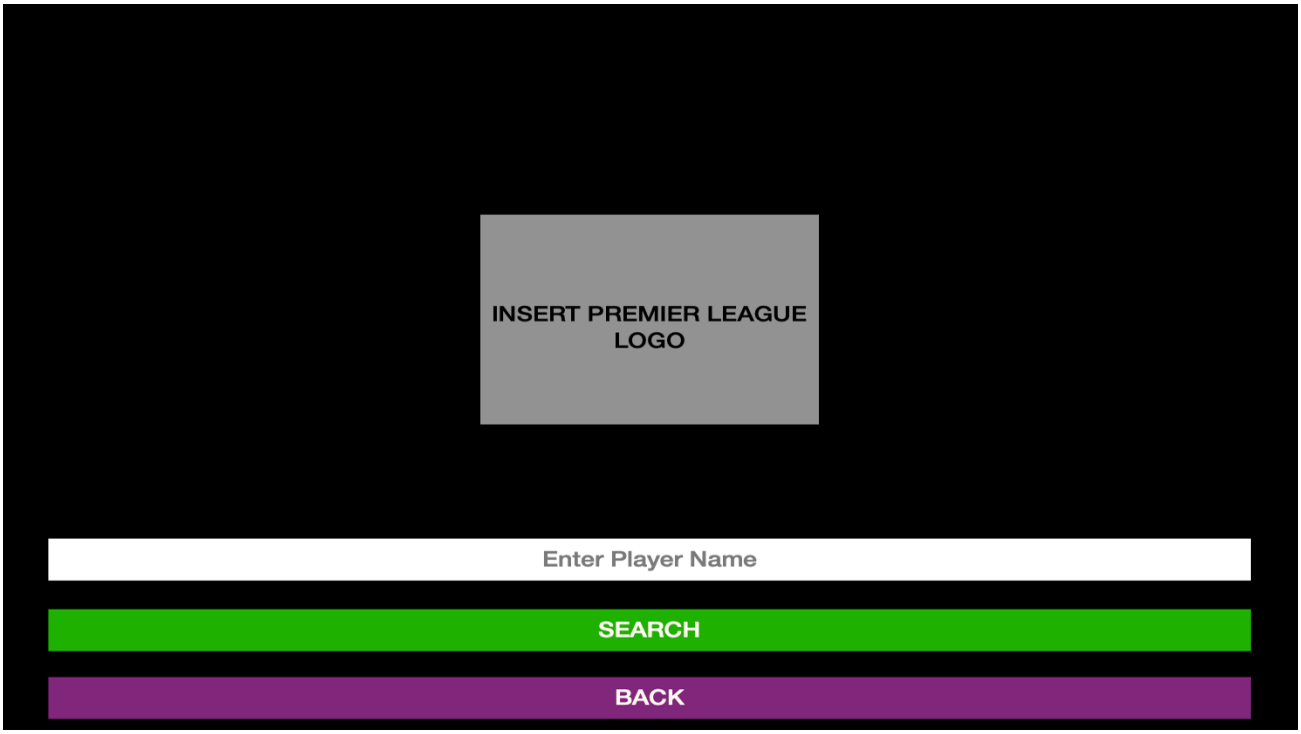


Final GUI

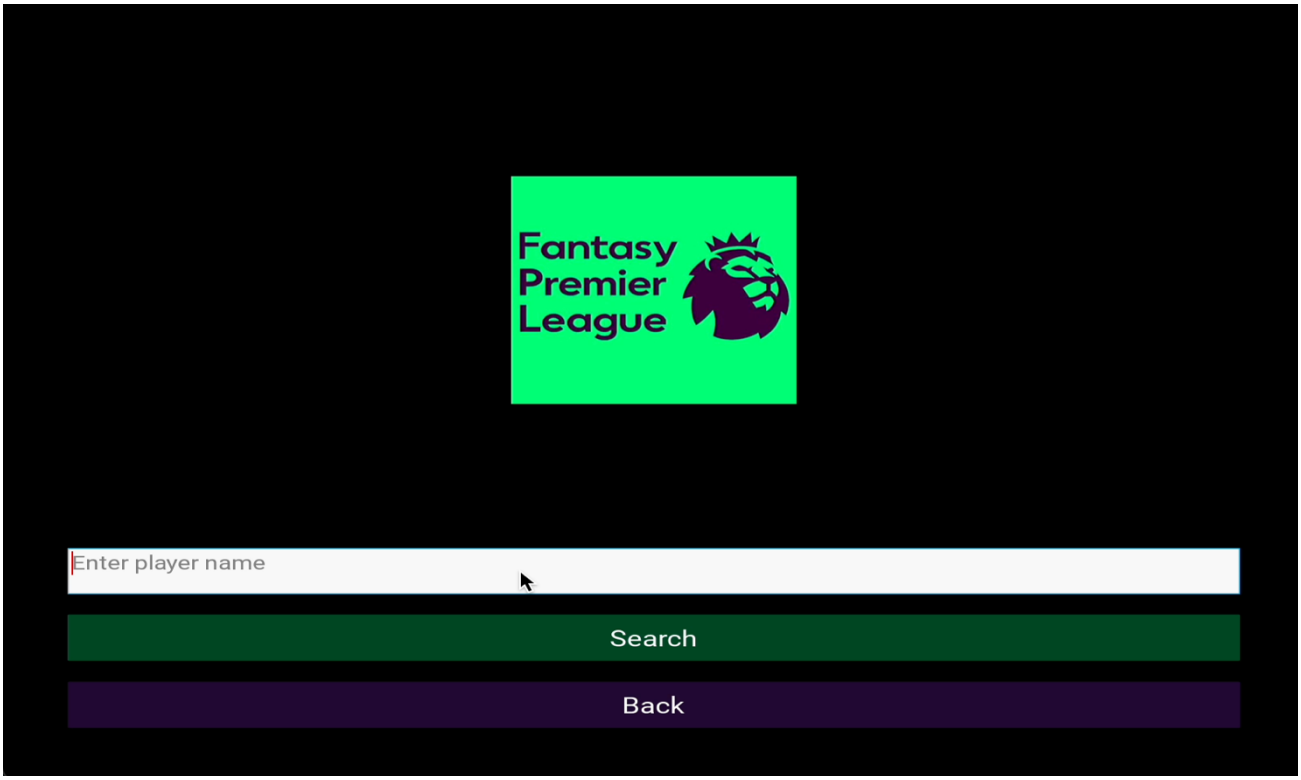


Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Design



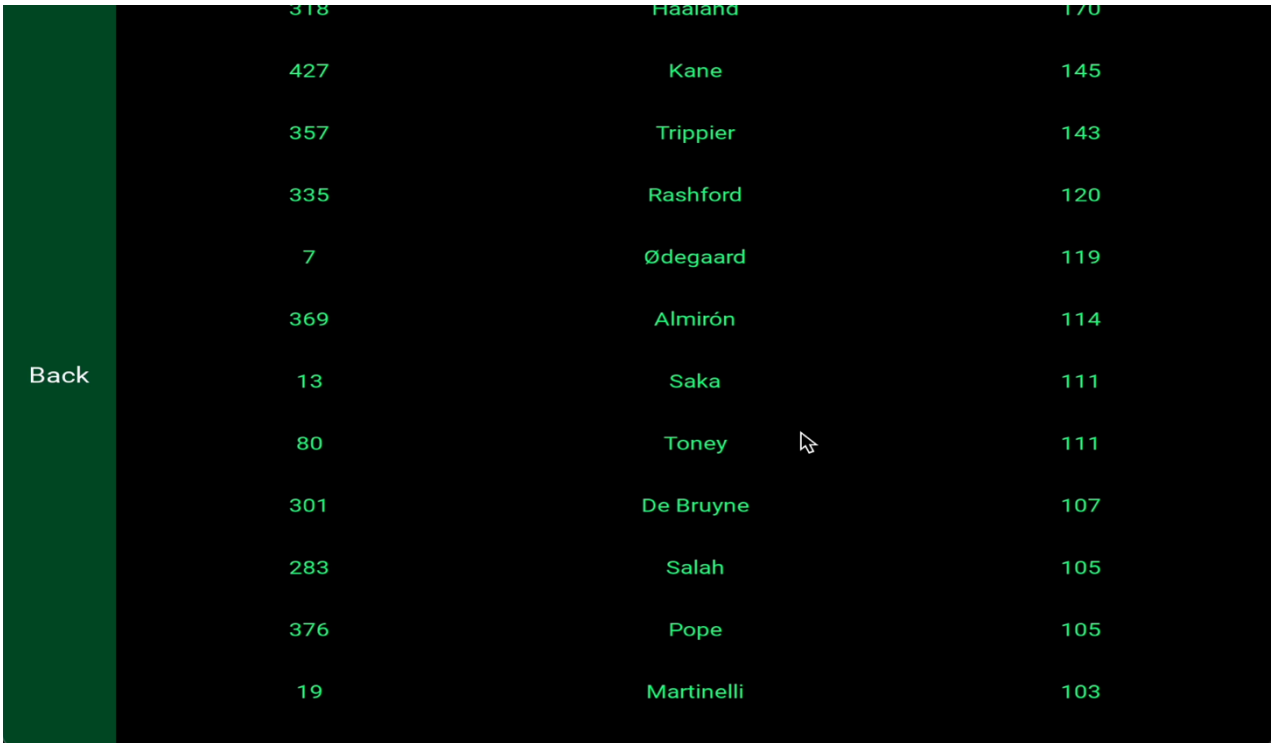
Final GUI



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Design

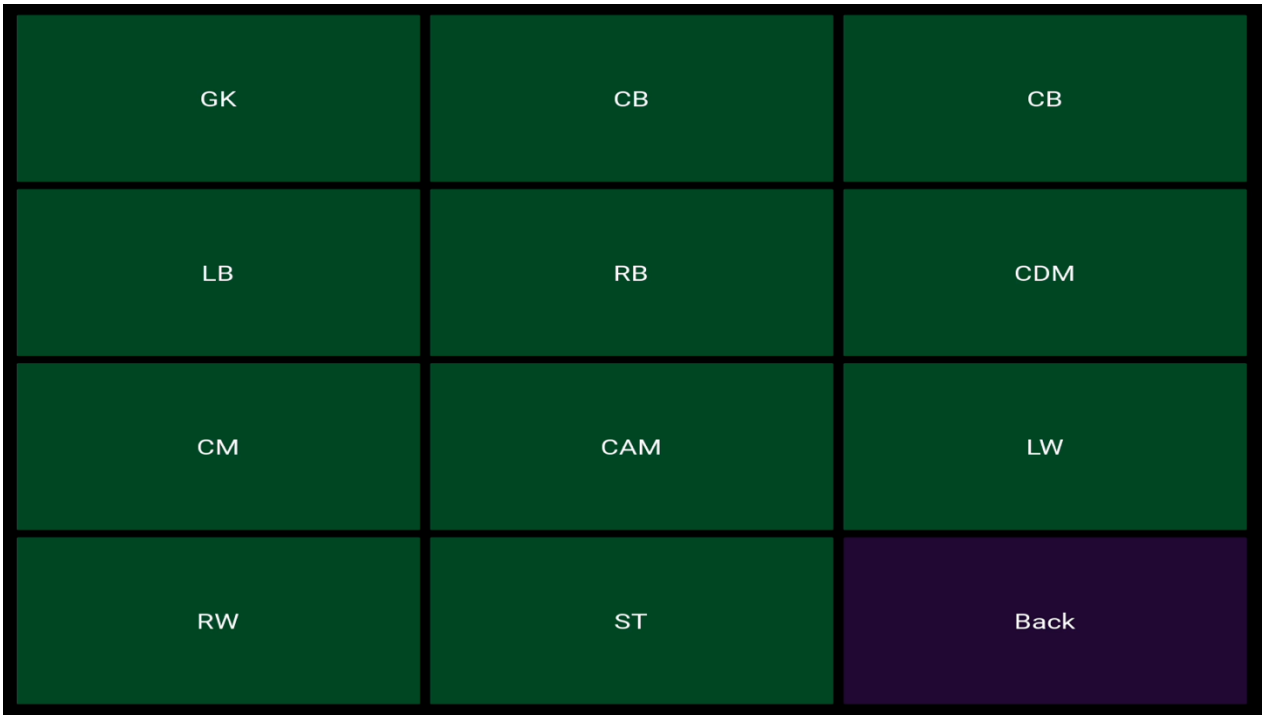
Final GUI



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

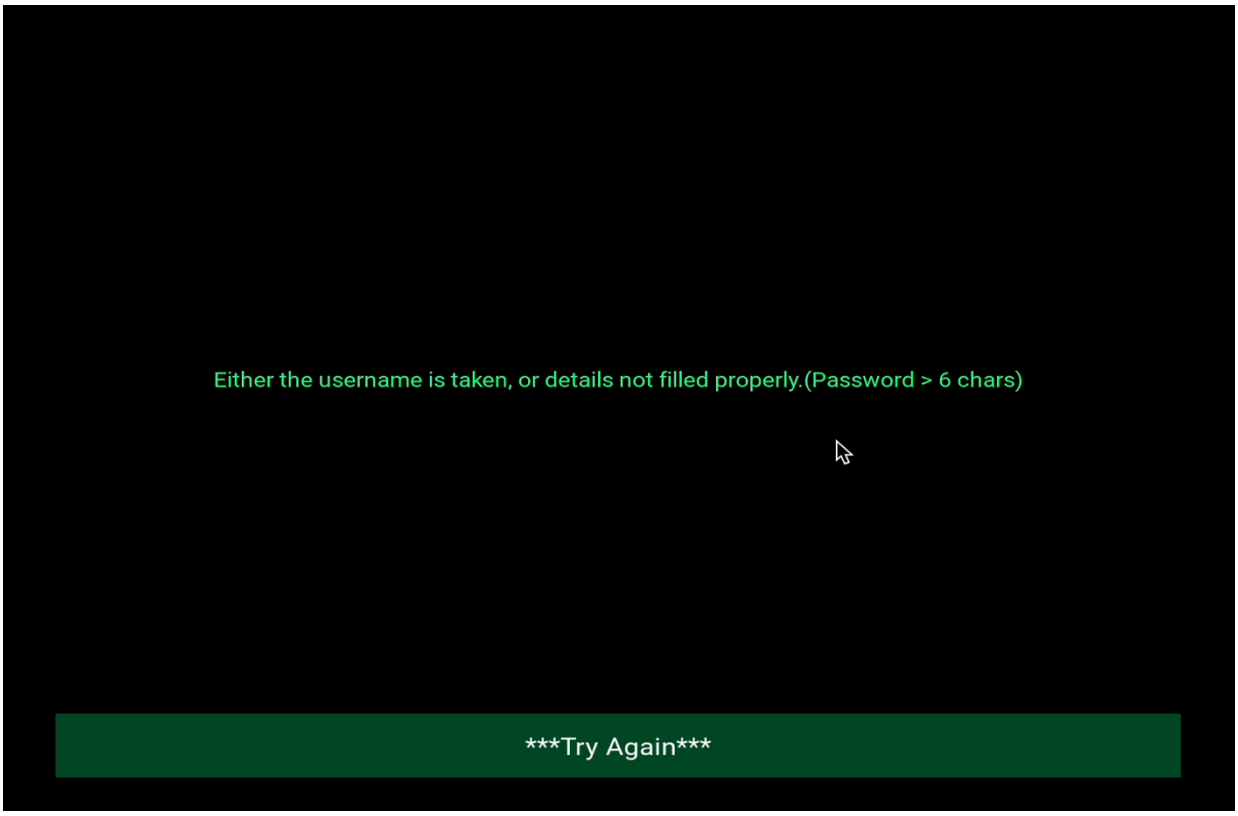
Design

Final GUI



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

As I mentioned, I also added error pages, which were not part of my initial design, one of them being an example below:



OBJECTIVE 5

MACHINE LEARNING MODEL – COLLECT DATA AND TRAIN A NEURAL NETWORK TO MAKE PREDICTIONS BASED ON NEW DATA ABOUT THE PREDICTED POINTS OF A FOOTBALLER

I decided to use Keras on TensorFlow for my machine learning model. As mentioned previously, I needed to collect data and convert it into a .csv file which I could train the model based on.

Achieved? ✅

I used Keras on TensorFlow to make my model. As mentioned in my video, I used over 15000 sets of data as mentioned in {Testing} to train my model off. To work out the best structure of my neural network, I had to do a lot of trial and error, I initially started off with 2 layers and 100 epochs with a batch size of 30. This ended up giving me a loss value of around 12. This was not a good enough value as loss is a value which

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

represents the summation of errors from your model. The lower this value, the better you model, however I did not want this value to be below 3, as this meant that I may be overfitting my data, which means I would be using a model, which would fit exactly against the training data hence would take values and results from that as an absolute, and not provide accurate results. Therefore finally, when I used 150 epochs and a batch size of 34 over three layers, I got a loss value of 4.1. This value meant I would get very good results and that they would be reasonably accurate.

Further Improvements considering client's feedback

My client's initial demands have all been met, however, I asked him what future upgrades my app could have, which would make it better.

The first suggestion is that users be allowed to create a team like they would in the real FPL game and that my app stores that team on their app. This could be easily done using the 'My Team' screen I have created, as well as the already functional account system I have. This new feature would display the user team and the predicted points of each player in that team, as well as the total predicted points. {APPENDIX}

Another update, that, in my opinion, could be done if this app is taken further in the future, is that I could add music in the background of the app, this could be the Premier League theme music. The reason I did not do this as an additional feature for my client in this version is that firstly, my client did not need that for this app and nor did he want it. Also, playing the actual them may lead to copyright issues unless in the future I get permission. However, for the purpose of fulfilling my client's needs, my NEA 'worked great for {him}' in the words of my client, Thomas Clark himself.{APPENDIX}

I also believe that to make my app more enjoyable, I could add different themes for my GUI, and give the user an option to choose between them. The GUI could also be made even more interesting if I made interactable objects such as player images and team logos which the user could click on. I believe this would make my app significantly more appealing and fun for a wider population of people.

Summary

Overall, I have ticked off every objective which I created in order to make the desired product for my end-users; I have also improved on them in my GUI and made a very good model, which will only get better with time. I have made a very functional and user-friendly product, which uses lots of the highest-level techniques in the back end and makes for an efficiently coded program.{APPENDIX}

However, as mentioned previously, my app is not perfect and can be improved. My main thoughts on my app are that once I can gather more data, over the period of this whole season, and more seasons to come, I could make my prediction model even better and this would lead to better predictions for my client. I believe this can naturally happen, if I re-train my model every 5 game weeks, and eventually the 15,000 sets of data will increase to 20,000 and so on.

In addition to this, even though my client is quite happy with the GUI and the overall appearance of the app, I feel that it can be made even more modern, and perhaps less 'clumsy'. To do this, I can add images of players as mentioned previously and create a better GUI for specific teams and their players. This would make my app more informative and enjoyable overall.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

REFERENCE LIST

#[1] – https://fplform.com/fpl-predicted-points

- http://www.fplstatistics.co.uk/Home/IndexAndroid

#[2] - https://medium.com/@frenzelts/fantasy-premier-league-api-endpoints-a-detailed-guide-acbd5598eb19

I used this endpoint information to collect my data from the Premier League API.

#[3] - https://docs.python.org/3/library/hashlib.html

#[4] - https://www.blog.pythonlibrary.org/2022/01/25/pysimplegui-an-intro-to-laying-out-elements/

#[5] - https://kivy.org/doc/stable/

I used the Kivy documentation for my technical solution as well as for my research.

#[6] - https://docs.python.org/3/library/tkinter.html

#[7] - https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

https://scikit-neuralnetwork.readthedocs.io/en/latest/

#[8] - https://realpython.com/python-ai-neural-network/

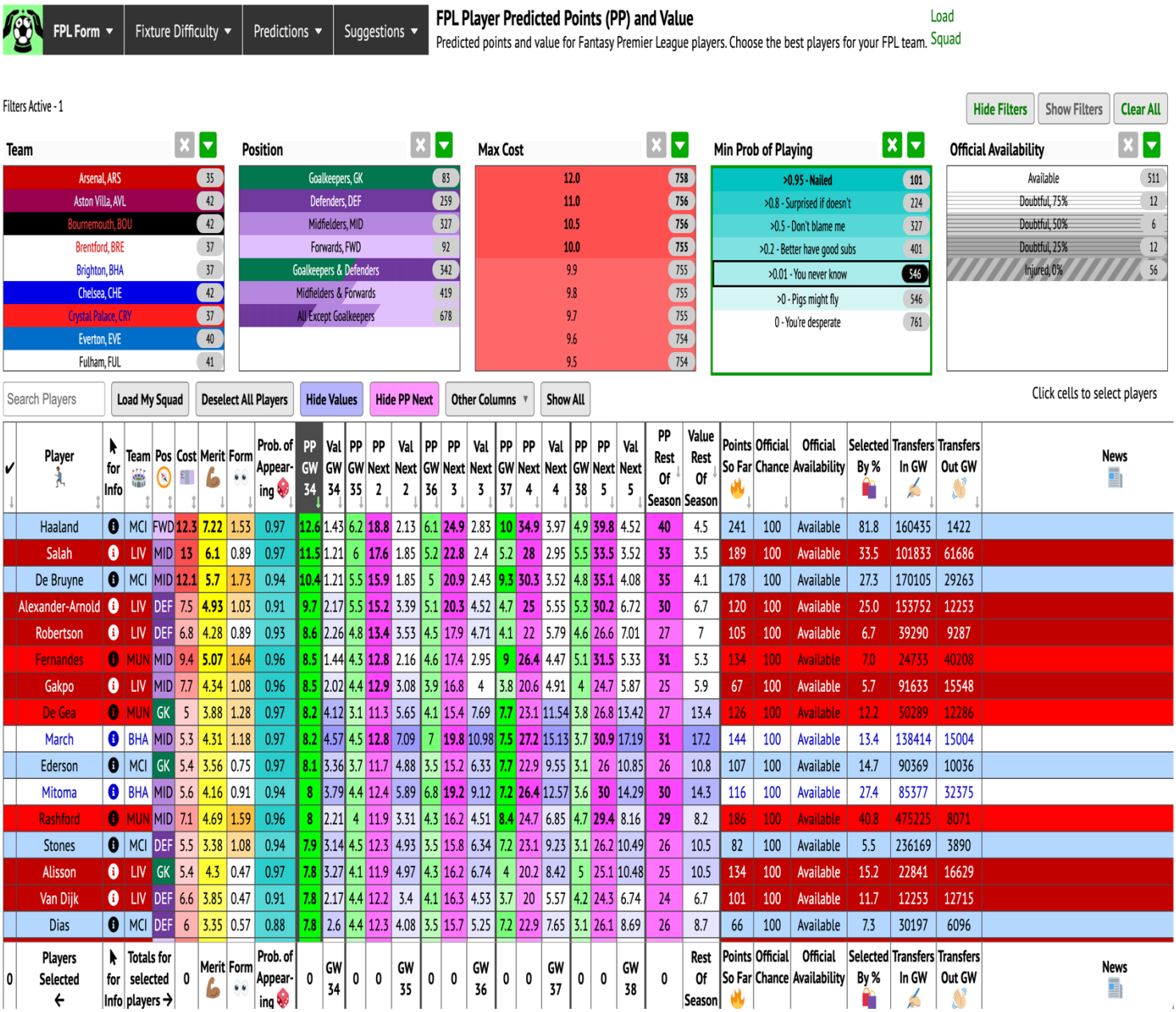
I used all of the links from 7 and 8 for my neural network implementation

APPENDIX

RESEARCH

As you have seen my reference list {REFERENCE LIST}, I did extensive amounts of research to ensure I could make the best product available for my client.

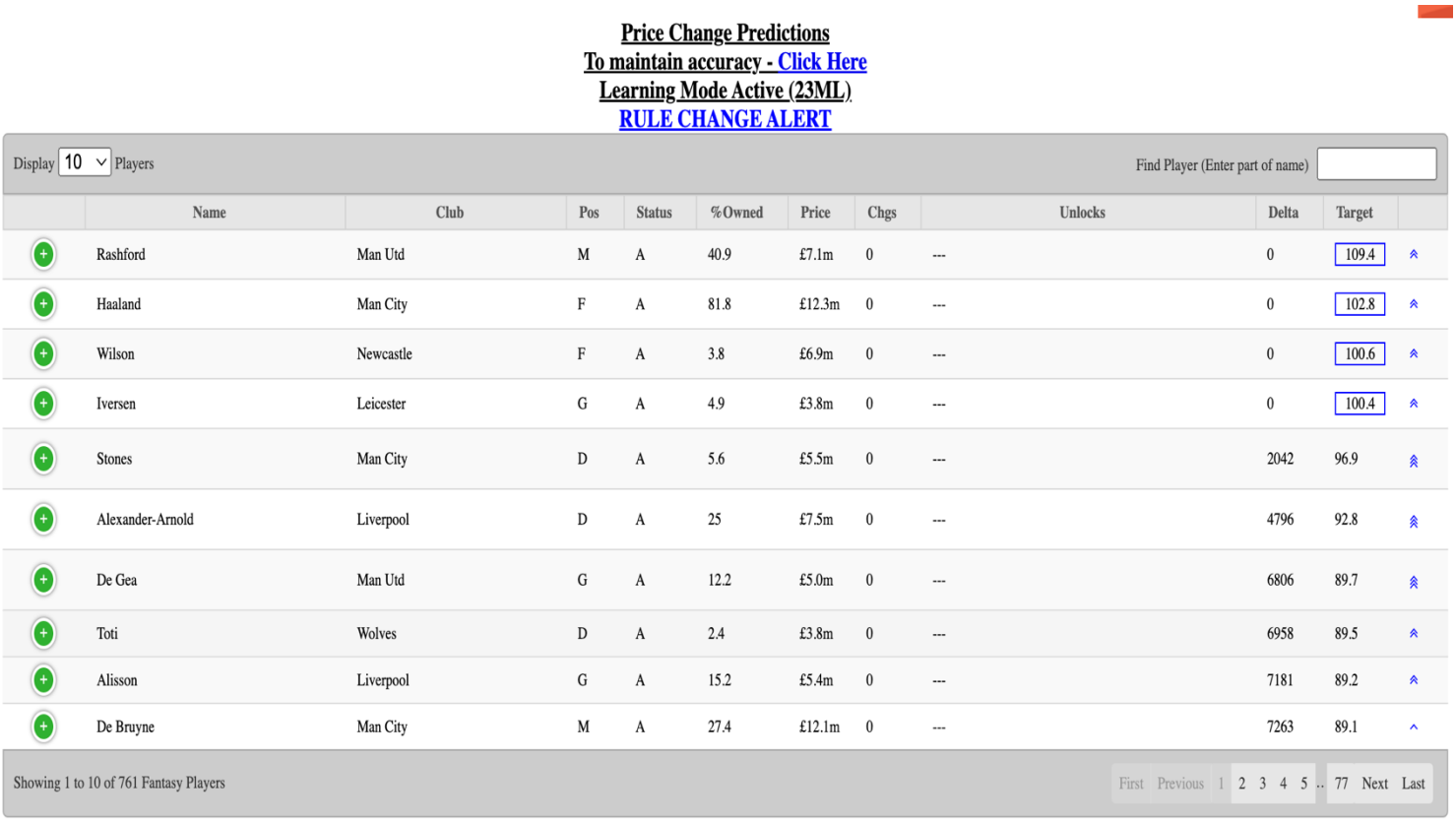
#1



✔	Player	for Info	Team	Pos	Cost	Merit	Form	Prob. of Appearing	PP GW 34	Val GW 34	PP GW 35	PP Next 2	Val Next 2	PP GW 36	PP Next 3	Val Next 3	PP GW 37	PP Next 4	Val Next 4	PP GW 38	PP Next 5	Val Next 5	PP Rest Of Season	Value Rest Of Season	Points So Far	Official Chance	Official Availability	Selected By %	Transfers In GW	Transfers Out GW	News
	Haaland		MCI	FWD	12.3	7.22	1.53	0.97	12.6	1.43	6.2	18.8	2.13	6.1	24.9	2.83	10	34.9	3.97	4.9	39.8	4.52	40	4.5	241	100	Available	81.8	160435	1422	
	Salah		LIV	MID	13	6.1	0.89	0.97	11.5	1.21	6	17.6	1.85	5.2	22.8	2.4	5.2	28	2.95	5.5	33.5	3.52	33	3.5	189	100	Available	33.5	101833	61686	
	De Bruyne		MCI	MID	12.1	5.7	1.73	0.94	10.4	1.21	5.5	15.9	1.85	5	20.9	2.43	9.3	30.3	3.52	4.8	35.1	4.08	35	4.1	178	100	Available	27.3	170105	29263	
	Alexander-Arnold		LIV	DEF	7.5	4.93	1.03	0.91	9.7	2.17	5.5	15.2	3.39	5.1	20.3	4.52	4.7	25	5.55	5.3	30.2	6.72	30	6.7	120	100	Available	25.0	153752	12253	
	Robertson		LIV	DEF	6.8	4.28	0.89	0.93	8.6	2.26	4.8	13.4	3.53	4.5	17.9	4.71	4.1	22	5.79	4.6	26.6	7.01	27	7	105	100	Available	6.7	39290	9287	
	Fernandes		MUN	MID	9.4	5.07	1.64	0.96	8.5	1.44	4.3	12.8	2.16	4.6	17.4	2.95	9	26.4	4.47	5.1	31.5	5.33	31	5.3	134	100	Available	7.0	24733	40208	
	Gakpo		LIV	MID	7.7	4.34	1.08	0.96	8.5	2.02	4.4	12.9	3.08	3.9	16.8	4	3.8	20.6	4.91	4	24.7	5.87	25	5.9	67	100	Available	5.7	91633	15548	
	De Gea		MUN	GK	5	3.88	1.28	0.97	8.2	4.12	3.1	11.3	5.65	4.1	15.4	7.69	7.7	23.1	11.54	3.8	26.8	13.42	27	13.4	126	100	Available	12.2	50289	12286	
	March		BHA	MID	5.3	4.31	1.18	0.97	8.2	4.57	4.5	12.8	7.09	7	19.8	10.98	7.5	27.2	15.13	3.7	30.9	17.19	31	17.2	144	100	Available	13.4	138414	15004	
	Ederson		MCI	GK	5.4	3.56	0.75	0.97	8.1	3.36	3.7	11.7	4.88	3.5	15.2	6.33	7.7	22.9	9.55	3.1	26	10.85	26	10.8	107	100	Available	14.7	90369	10036	
	Mitoma		BHA	MID	5.6	4.16	0.91	0.94	8	3.79	4.4	12.4	5.89	6.8	19.2	9.12	7.2	26.4	12.57	3.6	30	14.29	30	14.3	116	100	Available	27.4	85377	32375	
	Rashford		MUN	MID	7.1	4.69	1.59	0.96	8	2.21	4	11.9	3.31	4.3	16.2	4.51	8.4	24.7	6.85	4.7	29.4	8.16	29	8.2	186	100	Available	40.8	475225	8071	
	Stones		MCI	DEF	5.5	3.38	1.08	0.94	7.9	3.14	4.5	12.3	4.93	3.5	15.8	6.34	7.2	23.1	9.23	3.1	26.2	10.49	26	10.5	82	100	Available	5.5	236169	3890	
	Alisson		LIV	GK	5.4	4.3	0.47	0.97	7.8	3.27	4.1	11.9	4.97	4.3	16.2	6.74	4	20.2	8.42	5	25.1	10.48	25	10.5	134	100	Available	15.2	22841	16629	
	Van Dijk		LIV	DEF	6.6	3.85	0.47	0.91	7.8	2.17	4.4	12.2	3.4	4.1	16.3	4.53	3.7	20	5.57	4.2	24.3	6.74	24	6.7	101	100	Available	11.7	12253	12715	
	Dias		MCI	DEF	6	3.35	0.57	0.88	7.8	2.6	4.4	12.3	4.08	3.5	15.7	5.25	7.2	22.9	7.65	3.1	26.1	8.69	26	8.7	66	100	Available	7.3	30197	6096	
0	Players Selected ←	for Info	Totals for selected players →	0	Merit	Form	Prob. of Appearing	0	GW 34	0	0	GW 35	0	0	GW 36	0	0	GW 37	0	0	GW 38	0	Rest Of Season	Points So Far	Official Chance	Official Availability	Selected By %	Transfers In GW	Transfers Out GW	News

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

As you can see from the image on the previous page taken from the link provided in {REFERENCE LIST} (like all the images below), that my product had already been something people had tried to make. However, this website for instance, which was one that stood out to me during my research, provides the user with a mountain-load of information, however, it is very difficult to understand what any of this information means. As mentioned in my {ANALYSIS}, the overload of colours used doesn't contribute positively to the appearance and experience of this website. This is something I took inspiration from and ensured, as mentioned in {ANALYSIS, DOCUMENTED DESIGN, EVALUATION}, that the appearance of my app be appealing and not overloaded with bright colours for no apparent reason.



Price Change Predictions
To maintain accuracy - Click Here
Learning Mode Active (23ML)
RULE CHANGE ALERT

Display 10 Players — Find Player (Enter part of name)

	Name	Club	Pos	Status	%Owned	Price	Chgs	Unlocks	Delta	Target
+	Rashford	Man Utd	M	A	40.9	£7.1m	0	---	0	109.4
+	Haaland	Man City	F	A	81.8	£12.3m	0	---	0	102.8
+	Wilson	Newcastle	F	A	3.8	£6.9m	0	---	0	100.6
+	Iversen	Leicester	G	A	4.9	£3.8m	0	---	0	100.4
+	Stones	Man City	D	A	5.6	£5.5m	0	---	2042	96.9
+	Alexander-Arnold	Liverpool	D	A	25	£7.5m	0	---	4796	92.8
+	De Gea	Man Utd	G	A	12.2	£5.0m	0	---	6806	89.7
+	Toti	Wolves	D	A	2.4	£3.8m	0	---	6958	89.5
+	Alisson	Liverpool	G	A	15.2	£5.4m	0	---	7181	89.2
+	De Bruyne	Man City	M	A	27.4	£12.1m	0	---	7263	89.1

Showing 1 to 10 of 761 Fantasy Players — First Previous 1 2 3 4 5 .. 77 Next Last

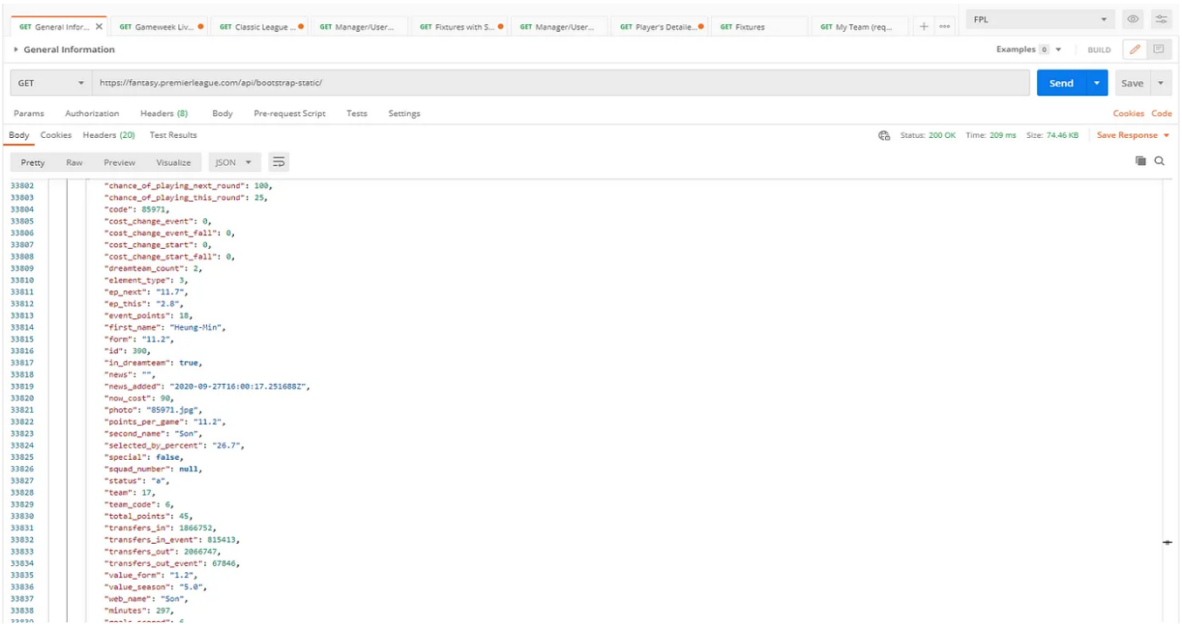
The second link to #[1] is the link that leads to website shown above. As I researched further to find a product, which had a more appealing interface, I stumbled upon this website. In my opinion, this is a better UI than the first website shown on the previous page. However, this also feels like it takes it too far to the other side to the point where there are no colours, which resonate with the FPL fans. Therefore, I tried finding a middle ground between these two products.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

#² –

Fantasy Premier League API Endpoints: A Detailed Guide

A detailed guide to all currently available Fantasy Premier League API endpoints.



We all know that Fantasy Premier League (FPL) is a game which is heavily based on data. Some of the you might also know that there is some API endpoints that we could use to get data about players, points, fixtures, leagues, etc. In this article I will explain about each endpoints I have used to get all those information.

I had to do an immense amount of research to in order to collect my data from my API, which is mentioned as an objective in {ANALYSIS}. To collect data from the Premier League API, I had to research about all the different URL's, which stored different types of data. For example, the image on the next page shows the 'bootstrap-static' URL:

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

1. General Information

Endpoint path: **bootstrap-static/**

Full URL: **https://fantasy.premierleague.com/api/bootstrap-static/**

JSON response of bootstrap-static endpoint

This endpoint returns general information about the FPL game divided into these sections:

- **events:** Basic information of every Gameweek such as average score, highest score, top scoring player, most captained, etc.

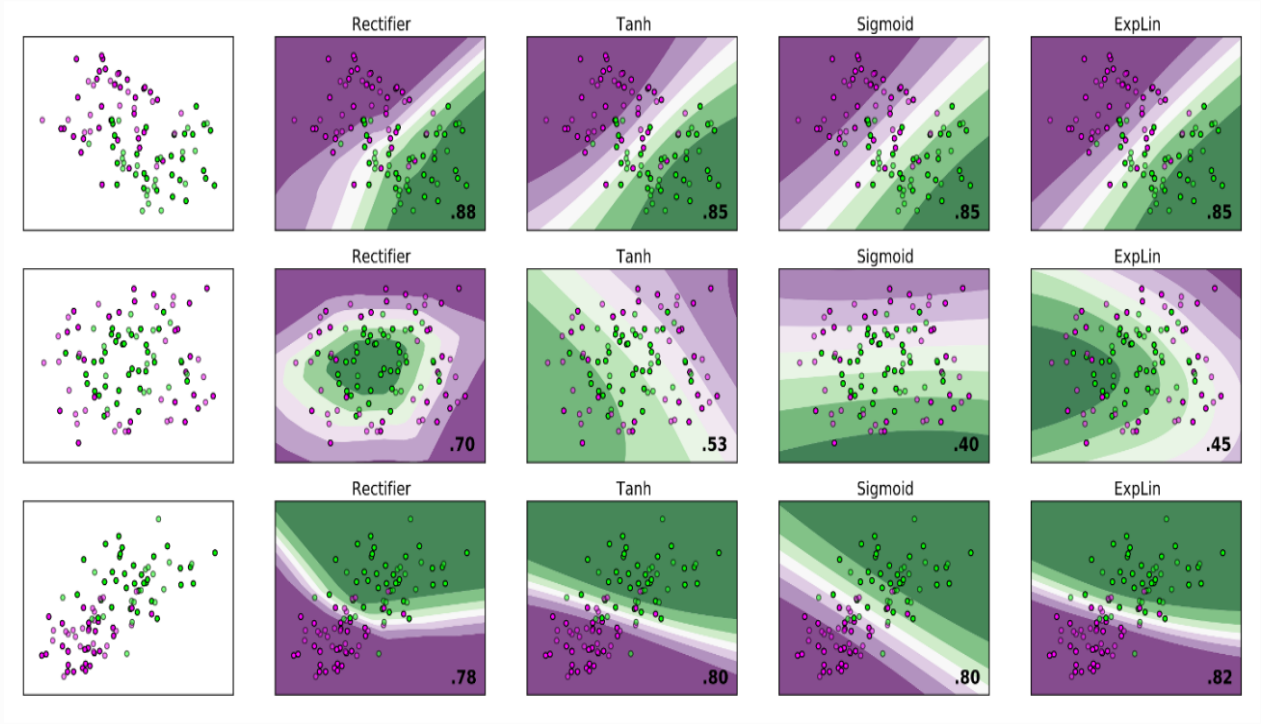
As you can see, this URL specifically provides data about the actual FPL game and the teams and players in the league.

#7 - To learn about neural networks I had to do research all over the internet, so I could learn how machine learning worked, what neural networks were, and how I could implement a neural network. My best choice turned out to be TensorFlow and Keras but on the next page is one of the websites I used to learn about different typed of rectifiers, backpropagation, and different activation functions, as talked about in the {ANALYSIS}.

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

Welcome to sknn's documentation!

Deep neural network implementation without the learning cliff! This library implements multi-layer perceptrons as a wrapper for the powerful `pylearn2` library that's compatible with `scikit-learn` for a more user-friendly and Pythonic interface.



Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

INITIAL INTERVIEW WITH THOMAS CLARK [CLIENT]:

WHAT IS THE PRODUCT THAT YOU WISH ME TO MAKE?

I PLAY THIS GAME CALLED FPL, IF YOU KNOW WHAT THAT IS, AND I WANT TO USE AN APP THAT PREDICTS HOW MANY POINTS AL THE PLAYERS ARE GOINGH TO GET EVERY WEEK SO I CAN CATCH UP WITH MY MATES IN MY LEAGUES.

WHAT IS IT THAT YOU ACTUALLY WANT FROM THIS APP?

I HAVE A FEW DEMANDS, FIRSTLY: I NEED IT TO BE A NICE-LOOKING APP THAT I CAN USE BUT IT MUST BE SIMPLE AND EASY TO USE, SO THAT WHEN I REGULARLY WANT TO CHECK THE PREDICTIONS OF PLAYERS, I CAN DO IT QUICKLY. SECONDLY: I NEED IT TO ACTUALLY DO THE MAIN THING OF PREDICTING THE POINTS OF EVERY PREMIER LEAGUE FOOTBALLER REASONABLY ACCURATELY. ALSO, I WOULD LIKE IT IF I COULD SEE THE TOTAL POINTS OF ALL THE PLAYERS FOR THE SEASON SO I CAN EASILY GET IDEAS FOR NEW PLAYERS I MAY WANT TO BUY THAT I HADN'T ALREADY THOUGHT OF.

WHAT IS THE BEST WAY OF GATHERING THE DATA, IN YOUR OPINION?

IN MY OPINION, THE BEST WAY TO GATHER THE DATA RELIABLY IS VIA THE PREMIER LEAGUE API AND THEN STORE THAT USING AN EASY AND PROTABLE DATABASE LIKE SOME FORMS OF SQL. THIS WILL ALLOW FOR EASY STORAGE AND WHEN DATA IS TO BE RETRIEVEDM THIS WILL BE DONE EFFICIENTLY, AND SECURELY ONCE YOU DO DEFENSIVE PROGRAMMING AND PREVENT SQL INJECTIONS.

WHAT ARE YOUR IDEAS ON THE PREDICTION PART OF THE PRODUCT?

THE BEST WAY TO DO THIS IS BY COLLECTING A LOT OF DATA USING THE PREMIER LEAGUE API, AND THEN CREATING A NEURAL NETWORK WHICH LEARNS FROM THIS DATA AND MAKES PREDICTIONS BASED ON THE NEXT WEEKS DATA THAT IT IS PROVIDED WITH.

ARE THERE ANY OTHER DEMANDS, ABOUT THE GUI OR ABOUT ANY FEATURES?

I WOULD LIKE IT IF THE APP'S COLOUR THEME WAS SIMILAR TO THE REAL PREMIER LEAGUE COLOURS, SO I WOULD LIKE IT TO BE A GREEN AND PURPLE THEME, AND IT WOULD ALSO BE NICE TO HAVE LOGOS OF THE

Candidate Name: Prannvat Singh
Candidate Number: 6322
Centre Name: Maiden Erlegh School
Centre Number: 51603

PREMIER LEAGUE IN THE GUI TO MAKE IT MORE INTERESTING.

FINAL INTERVIEW WITH THOMAS CLARK [CLIENT]:

HOW WELL WOULD YOU SAY HAVE I CREATED AN APP TO THE STANDARD THAT YOU INITIALLY WISHED?

HONESTLY, THIS APP HAS EXCEEDED MY EXPECTATION. WHEN YOU FIRST SHOWED ME THE DESIGNS FOR THE SCREENS, I THOUGHT THEY WERE JUST WHAT I WANTED – SIMPLE, YET APPEALING, AND THE FINAL INTERFACE IS EVEN BETTER THAN THE DESIGN BECAUSE YOU TONED DOWN THE COLOURS AND MADE IT A MORE RELAXED YET INTERESTING DESIGN TO LOOK AT. THE ACCESSIBILTY OF EVERYTHING FROM THE MAIN MENU IS AMAZING AND SIMPLE TO USE. ALSO, THE ACCOUNT SYSTEM IS A GREAT FEATURE AND YOU CAN USE THIS FOR FURTHER UPDATES AND INTRODUCE NEW FEATURES TO YOUR APP TOO. THE MAIN THING , WHICH WAS THE ACCURACY OF THE PREDICTIONS HAS GENUINELY SURPRISED ME. I CHECKED SO MANY PLAYERS OVER THE PAST COUPLE OF WEEKS AND MOST OF THEM HAVE BEEN GETTING POIINTS IN A VERY SIMILAR RANGE TO WHAT THE APP PREDICTED. FOR EXAMPLE, THE OTHER DAY, THE APP PREDICTED THAT HAALAND WOULD GET 8 POINTS, SO I BOUGHT HIM, AND HE ENDED UP GETTING 10 POINTS, SO THAT WORKED GREAT FOR ME!

WHAT ARE POSSIBLE UPDATES AND IMPROVEMENTS I COULD MAKE TO THIS APP?

IN MY OPINION, THE MAIN IDEA I HAVE FOR POSSIBLE IMPROVEMENTS IS THAT YOU CAN USE THE ACCOUNT SYSTEM YOU HAVE MADE AND THE 'MY TEAM' SCREEN, WHICH HAS ALL THE POSITIONS ON IT TO MAKE A FEATURE IN WHICH USERS ARE ABLE TO CREATE THEIR TEAM BY CHOOSING PLAYERS IN EACH POSITION, AND THEN THE APP SAVES THIS TEAM AND GIVES A TOTAL PREDICTION OF THE WHOLE TEAM, AND EVERYTME A USER LOGS IN, THEY CAN EDIT THEIR TEAM OR JUST VIEW THEIR TEAM AND SEE THE PREDICTED POINTS. I THINK THAT WOULD BE A FUN AND USEFUL FUTURE UPDATE TO THIS PRODUCT.