Reza Talakoob - Pooya Khoshabi
40231714 - 40163944
rezatalakoob22@gmail.com -
pouya.khooshabi@gmail.com

Project Report - SOEN 6441

Access to Github

You can access the Github Repository by this link: github-link

1 Introduction

Fantasy Premier League is an online game in which one takes on the role of a Premier League player's fantasy manager. In this game, users receive points based on how football players perform each week. Players select a virtual squad of Premier League players, and if they perform well, they achieve points.
FPL is a game that is heavily data-driven, and one can access specific API endpoints to obtain information on players, points, schedules, leagues, and other topics.
In this work, we will analyze players' statistics and recommend a combination of 15 players to the user. The recommended team consists of the players in their best shape, and this will help users reconstruct their current team to get suitable players with the maximum budget they have.

2 Dataset

We read the API from this link and will use sqlite database to save them. We have three different tables in this work which we explain them in this section.

2.1 Table of Teams

This table contains information about the different teams in the premier league. Every player belongs to a team and a team can have one or more players from a certain position. This table also contain the information about win, lose or draw and the number of played games by the specific team.

2.2 Table of Players

This table contains information about players. Each player belongs to one team, and each player also belongs to one position. This table also has information about the player's cost, form, total gained point and total minutes played by the player. A part of players' table is shown in figure 1.

	id	second_name	team	element_type	selected_by_percent	chance_of_playing_next_round	value_season	now_cost	minutes	total_points
0	1	Alves Soares	1	2	0.1	100.0	0.2	42	27	1
1	3	Xhaka	1	3	4.0	NaN	11.4	51	1061	58
2	4	Elneny	1	3	0.7	100.0	0.5	42	90	2
3	5	Holding	1	2	0.1	NaN	0.7	42	11	3
4	6	Partey	1	3	0.5	100.0	7.7	48	779	37
...	...	...	...	...	...	...	...	...	...	...
445	579	Ganchinho Guedes	20	3	0.2	NaN	4.4	59	596	26
446	589	Nunes	20	3	0.2	50.0	5.8	48	848	28
447	608	Kalajdzic	20	4	0.0	0.0	0.2	53	45	1
448	625	Da Silva Costa	20	4	0.3	0.0	1.3	55	412	7
449	629	Traoré	20	3	0.1	100.0	2.4	45	290	11

450 rows × 10 columns

Figure 1: Table of Players

2.3 Table of Positions

This table contains information about the different positions we have in the game. Players will be selected based on these positions and get leveraged in the squad. A part of positions' table is shown in figure 2.

	id	plural_name	singular_name_short	squad_select	squad_min_play	squad_max_play	element_count
0	1	Goalkeepers	GKP	2	1	1	70
1	2	Defenders	DEF	5	3	5	233
2	3	Midfielders	MID	5	2	5	283
3	4	Forwards	FWD	3	1	3	73

Figure 2: Table of Positions

3 Implementation

Based on the requirements, we decided to implement two different application for this project. We can categorized our implementation in two categories. The first one take a look at the analysis with Object-relational pattern and the second one will solve the problem with Data Source Structure.

3.1 Object-Relational Pattern

In our Project, we look at each table in an Object Oriented fashion and try to model the problem with Object Relational Structure. Accordingly, we create Teams, Players, and Position classes.
Classes encapsulate the concept of each table where the table's name is the name of the class, and each column is an attribute of the corresponding class. Then, each row of the tables will be one object, and we can convert the whole table to arrays of objects.
What we implemented as the result of this structural pattern is to find how many teams and positions contain high-value players. High-value players are the ones where the ratio of total points over the cost for them is more than 10. Then we output the name of the teams and the positions in addition to the number of high-value players in the league. This helps a user improve their decision-making by picking players from a specific team or a position.
There is a $many - to - many$ relation between the team and position class where each team can belong to many positions, and each position can belong to many teams. The third class will address this problem and create a $one - to - many$ relation between the previous classes and itself. In other words, each player belongs to one team, and also each player belongs to one position. At the same time, one team and position can have many players.

3.2 Data Source Pattern

we store the 3 mentioned tables and their relations in a relational database called fpl-final.db. knowing that we have a many-to-many table mapping in our implementation(which uses a third table called players to do the mapping), we query the database with two nested joins to get the final players-stats dataframe. we use this dataframe as input for our application.
In the $mainapp.py$ module, we define some functions that use this players-stats dataframe. Then we group players based on their position and sort them by their value-season(which is total points divided by cost.) we finally use these sorted position-based dataframes to make our final team sheet. We start with 2 Goalkeepers and then pick other players one by one from each position knowing that we cannot use more than 3 players from a specific pl team. we need exactly 5 Defenders, 5 Midfielders, and 3 forwards.

4 Main Criteria

This section covers the main requirements for the project. These criteria are discussed below.

4.1 Coding standard

The coding standard which we are using in this project is PEP8. The detail of this standard are discussed here. In order to apply this standard in our implementation we use Pylint framework. Pylint is a python library which corrects the code based on PEP8 standard and gives feedback and rate to the code. We leveraged this feature and unified our code in this standard. Figure 3 is the rate of pylint for one of the python files called $team.py$.
Suggestions were mainly about providing explanations for the module, classes, and functions/methods. Also

suitable naming styles for the variables and constants and keeping lines short are some of the most common suggestions among others.

```
pooya@Pooyas-MacBook-Air Project % pylint team.py
************* Module team
team.py:2:0: R0902: Too many instance attributes (9/7) (too-many-instance-attributes)
team.py:7:4: R0913: Too many arguments (7/5) (too-many-arguments)

-------------------------------------------------------------------
Your code has been rated at 8.82/10 (previous run: 8.82/10, +0.00)
```

Figure 3: The view from applying Pylint on the code and its recommendations and rate

4.2 Applicable Pattern

As we already fully discussed above, two main Design patterns have been deployed for this project.(Please check 3 for full details)

4.2.1 Object-Relational Pattern

we try to model our problem with Object Relational Structure. We have 3 classes called Teams, Players, and Positions. This is done by mapping each column of our tables to their related class attributes. Two attributes are added to the Players class which models the composition relation between the player object and Positions/Teams objects. This object-relational model then has been used to obtain high-value players from each of the 4 playing positions.

4.2.2 Data Source pattern

we use table data gateway in order to obtain data from database. for this pattern, we query the database and store 3 tables in 3 dataframes. Then we use a nested join query to obtain the $players - stats$ dataframe. This has been done by taking into account the relations between our tables. Next, we use this dataframe as input for our team selection functions.

4.3 Refactoring

The refactoring strategy we are using is "Replace Array with Object". In the database, we have tables with different rows, and each row has different columns. For example, the table called "$slim\_elements\_types\_df$" contains columns called "id", "$plural\_name$", "$singular\_name$", etc. In our code, we encapsulated the columns as different attributes of the class "position". We also had a table called "$slim\_teams\_df$", which contains the details regarding each team. We encapsulated each column of this table as different attributes of the class called "$team$". We did the same for a table called "$slim\_elements\_df$" and saved the attributes in a class named "$player$". In this way, each row of the table will convert to a different object; instead of having a table, we only have a list of objects.

4.4 Testing tool

for the matter of testing, we use unit testing in python(the unittest tool). In this module, we define a class inherited from $unittest.TestCase$ which implements three methods for testing our main application, as there are three constraints we want to confirm on our final team sheet. first is to confirm that our team is well within the 1000 declared budget. Secondly, our final team should consist of exactly 15 players. and finally, we should assert that we do not have more than 3 players from any premier league side. By running $test_mainapp.py$ we can check that our output is in line with these rules.

4.5 Architecture of the Design

Based on our two different design patterns, we provide two architectures of the design in this section.
First, we take a look at the classes that we have. In this work, we have three classes: team, player, and position. The relation of classes is fully discussed in section 3.1. Figure 5 is the architecture of this pattern. In this Diagram, we can see how the classes are interconnected with the associations.
figure 5 shows how we retrieve data from the database using a table data gateway. This enables us to query our database in the data source layer and store the results in our domain objects which then can be used for our further analysis and representation.

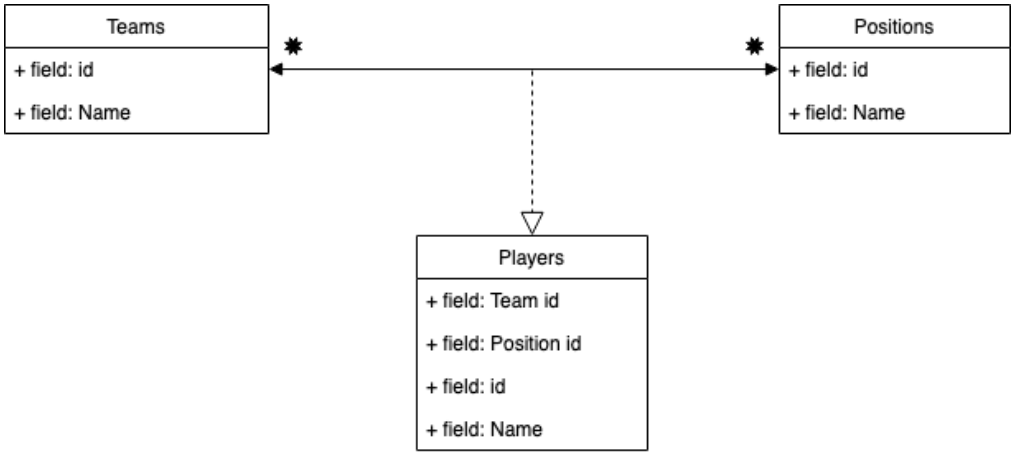

Figure 4: Architecture of Object relational pattern

5 Result

In this section, we show the result for both of the applications.
The first application solves the problem of finding the number of high-value players for each team and position. In figure 6 and figure 7, we can see that the number of high-value players from the Arsenal team is six, and the number of high-value players in the goalkeeper position is fourteen.
The second application solves the problem of recommending a team of 15 players to the user based on the form and cost of the players. The output may be different based on players' performance each week. Figure 8 shows the output for the date we implemented the algorithm.
As seen from the figures, the outputs consist of comprehensive recommendations based on the statistics of online data being updated every week. The purpose of both applications is to boost the user's decision-making in terms of players, teams, and positions and make them choose the best players in the league and, at the same time, balance the costs and save for future changes in their team.

References

1. https://medium.com/@frenzelts/fantasy-premier-league-api-endpoints-a-detailed-guide-acbd5598eb19
2. https://peps.python.org/pep-0008/
3. https://towardsdatascience.com/fantasy-premier-league-value-analysis-python-tutorial-using-the-fpl-api-8031edfe99108ef2
4. https://fpl.readthedocs.io/en/latest/classes

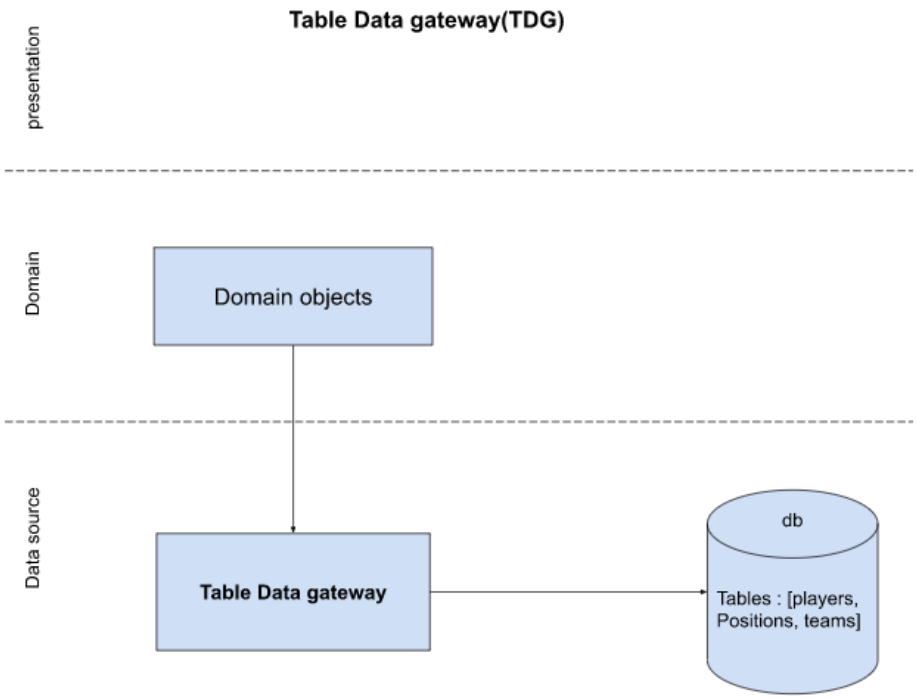

Figure 5: Table Data Gateway

```
Midfielders :  17
Defenders :  13
Goalkeepers :  14
Forwards :  2
```

Figure 6: number of high-value players for positions

```
Arsenal :  6
Aston Villa :  2
Bournemouth :  2
Brentford :  3
Brighton :  3
Crystal Palace :  2
Everton :  3
Fulham :  1
Leicester :  4
Leeds :  1
Liverpool :  1
Man City :  2
Man Utd :  1
Newcastle :  7
Nott'm Forest :  1
Spurs :  3
West Ham :  2
Wolves :  2
```

Figure 7: number of high-value player for teams

	id	second_name	selected_by_percent	chance_of_playing_next_round	value_season	now_cost	minutes	total_points	position	teams
0	398	Henderson	12.6	NaN	13.0	47	1260	61	Goalkeepers	Nott'm Forest
1	81	Raya Martin	8.3	NaN	12.6	46	1260	58	Goalkeepers	Brentford
2	258	Castagne	6.8	100.0	12.6	47	1101	59	Defenders	Leicester
3	366	Schär	11.5	100.0	11.8	49	1064	58	Defenders	Newcastle
4	10	White	8.4	NaN	11.5	46	970	53	Defenders	Arsenal
5	26	Saliba	32.7	NaN	11.5	52	1065	60	Defenders	Arsenal
6	526	Mee	5.0	NaN	11.3	46	1118	52	Defenders	Brentford
7	346	Hoelgebaum Pereira	24.4	100.0	12.8	46	1150	59	Midfielders	Fulham
8	104	Groß	13.8	NaN	12.1	56	1102	68	Midfielders	Brighton
9	3	Xhaka	4.0	NaN	11.4	51	1061	58	Midfielders	Arsenal
10	111	Trossard	26.1	NaN	11.1	70	1085	78	Midfielders	Brighton
11	70	Billing	1.1	NaN	10.6	53	1046	56	Midfielders	Bournemouth
12	80	Toney	15.3	100.0	9.5	74	1170	70	Forwards	Brentford
13	66	Solanke	4.4	100.0	9.1	57	985	52	Forwards	Bournemouth
14	210	Mitrović	26.6	75.0	8.8	69	1019	61	Forwards	Fulham

Figure 8: The recommended team