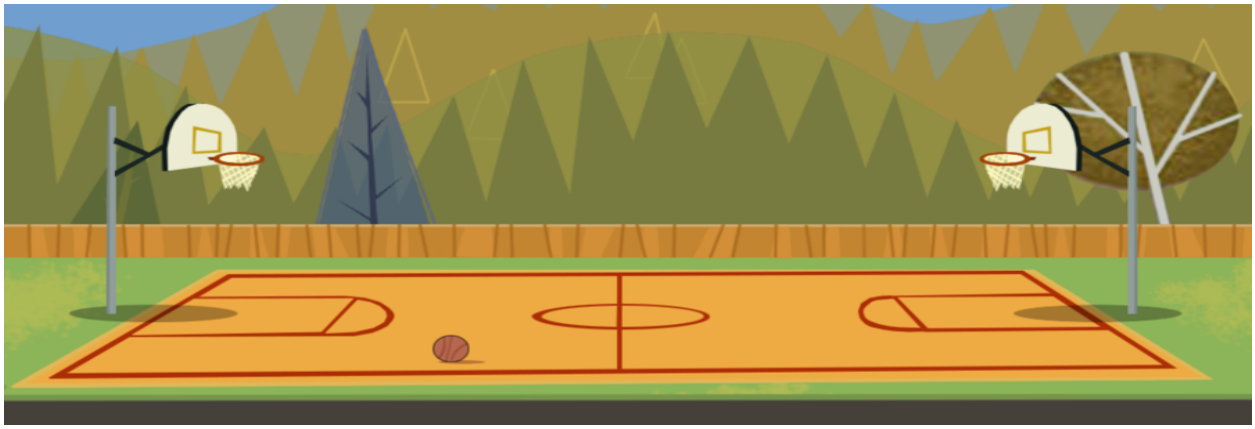
CS 411: Avinda De Silva, Abhishek (AJ) Jariwala, Nithin Nathan

NBA For Dummies

Stats Analysis of NBA Players



Introduction:

Since the beginning of the NBA, avid basketball fans have always had friendly debates over various basketball topics. Who was the best player in the 90's? Which championship team had the player with the most points? Who's the real G.O.A.T., Michael Jordan or Lebron James? Using the most up-to-date data from the NBA, our project offers accurate predictions to help determine the answers to these questions and allow these diehard fans to look at their favorite players in ways they've never seen before.

Details:

Purpose:

The purpose of "NBA For Dummies" is to provide an interactive game simulation web application for the NBA based on player preferences of the user. In other words, this app allows the user to determine what team lineup is the "best" from their knowledge of the NBA. This allows for users to search for specific players they want from a specific season and have the ability to add them to their team which can be viewed as a table on the

homepage. However, if some player stats are incorrect/missing for a player the user has the ability to update these stats manually. Moreover, the user also has the ability to delete certain players from their team if they choose to build a completely different lineup. Finally, once both of the users have built their lineups, the users can run the game simulator which will output the probabilities of that certain team winning over the other.

Functionalities:

- Search for any player in the NBA from 1946-2020
- Find the teammates of players
- Find similar players that have similar stats to a certain player
- Insert any player into the User 1 or User 2 databases (displayed on homepage)
- Update any stat of the players that have been added to the User team databases
- Delete any player from the User team databases
- Run a game simulation based on User team databases to predict the winner

Basic Functions (Read):

Our project implemented all CRUD functionalities. For read, the search bar that we implemented takes in the search term, queries the database for the players we want, and displays the results onto the HTML webpage.

Advanced Function (Game Simulator):

For our AF1, we decided to come up with an algorithm that would determine which of two lineups of players was the best. The algorithm was complex because it took into account a variety of different factors and teams size differences into account when determining the winner. Additionally, we consider it advance because instead of just querying the different records that are found in the database, we use the different values/stats that are given for a player to our benefit and calculate an overall probability using a gaussian distribution along the values for each user-generated team. In general, we thought this advanced function would overall be best for our project because as mentioned before, there are several times when individuals debate best teams based on different categories in the NBA. By allowing fans to have a physical way to see which team would win rather than arguing with a friend, gives them a satisfying way to see the real winner.

Division of Labor:

Members	Front End	Back End	Database	Research	Testing
Avinda	X		X	X	X
Nithin	X	X		X	X
AJ		X	X	X	X

Data:

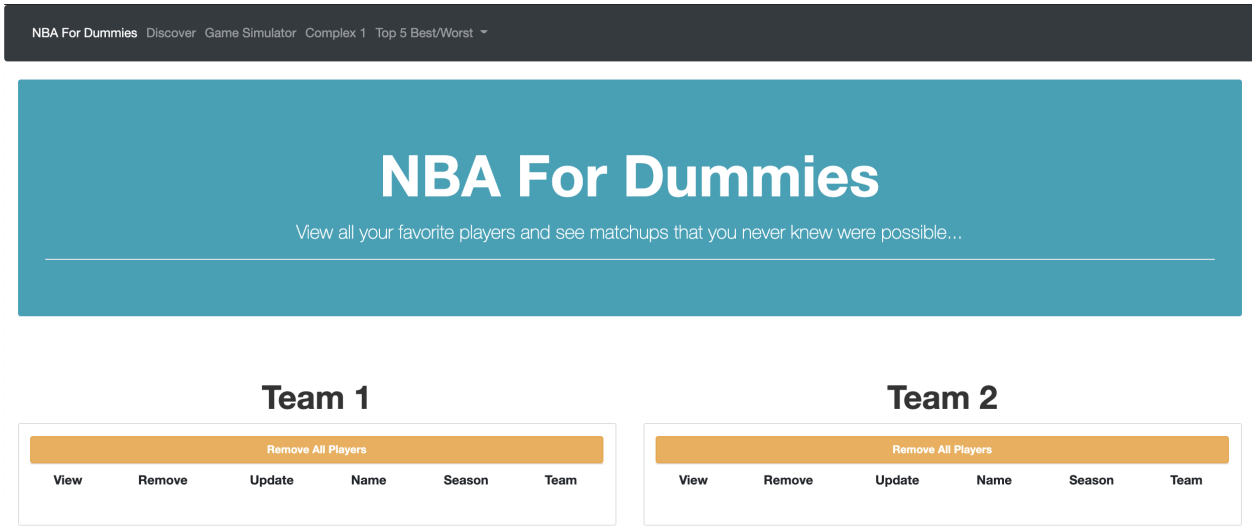
This application had four different databases. One of the databases was a Players database which consists of all of the players in the NBA from 1946-2020 and all of the seasons that each of these players have played in. For example, our Players database contains 17 instances of Lebron James which corresponds to all of the seasons he has been in the NBA for. Moreover, this application has a Teams database with basic team information such as: team names, location, when it was created, and more. Furthermore, this application has a Stats database with basic player stats such as: points, assists, rebounds, steals, field goal percentages, free throw percentages, and more. Finally, this application has a Games database with all of the games a team has played in a season with team stats for each of these games.

Data Collection:

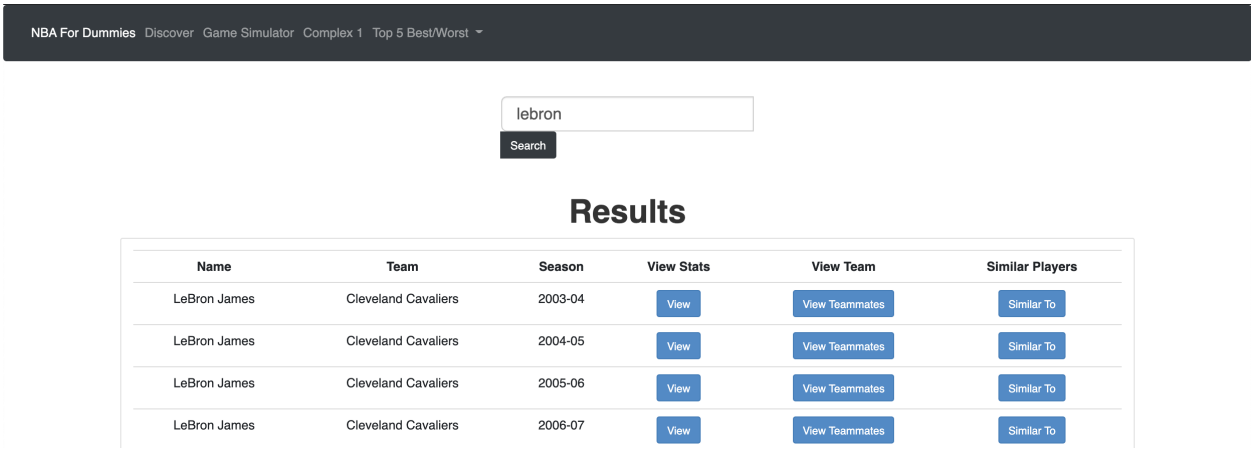
We collected our data from the nba_api, which is a python library that is connected to a NBA database through api calls. This api contained numerous endpoints containing data for all players in the NBA from 1946-2020, as well as Team Data, Play-by-Play, as well as data for each individual game. We crawled data from this api using a python script that was connected to our MySQL workbench, and imported data from the api into the different tables of our SQL Database. Our database contains records of over 4000 NBA players, as well as their stats, as well as Over 88000 records for individual game data.

Data Flow:

When a user starts up our application, they will be directed to a homepage containing two empty tables, these tables refer to the User's created teams and a navigation bar with different ways of representing the NBA data as well as the ability to view all NBA players from 1946-2020.



If a player clicks on the discover tab, they'll be directed to a page with a search bar that the user can then enter the name of a player of their liking. Once they click enter, search results will be shown of common player info for every player name that fits that query.



The user can then view stats of the player, by clicking on 'View' next to the player record, can view the player's teammates for that season, by clicking on 'View Teammates next to

the player record, and finally also has the ability to find similar players, by clicking on “Similar” next to the player record.

If the user chooses to view stats, they will then have the option to add that specific player to one of the user defined teams, these user-defined teams will be used for simulating games between the two teams.

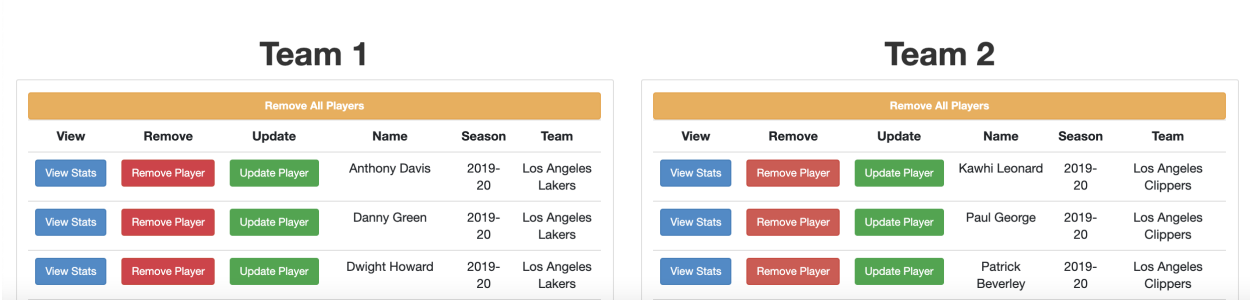
Stats Of LeBron James During 2003-04

Age	Games Played	Games Started	Minutes Played	Field Goals Made	Field Goal Attempts	Field Goal Percentage	3-Point Field Goals Made	3-Point Field Goal Attempts	3-Point Field Goal Percentage
19	79	79	3120	622	1492	0.417	63	217	0.29

Free Throws Made	Free Throw Attempts	Free Throw Percentage	Offensive Rebounds	Defensive Rebounds	Rebounds	Assists	Steals	Blocks	Turnovers	Personal Fouls	Points
347	460	0.754	99	333	432	465	130	58	273	149	1654

Add To Team 1 | Add To Team 2

After the player has been added to the team, the user can repeat this process until they have created two teams to their liking. They then have the ability to run a game simulator to demonstrate which team has the higher chance of winning this matchup.



Team 1

Remove All Players

View	Remove	Update	Name	Season	Team
View Stats	Remove Player	Update Player	Anthony Davis	2019-20	Los Angeles Lakers
View Stats	Remove Player	Update Player	Danny Green	2019-20	Los Angeles Lakers
View Stats	Remove Player	Update Player	Dwight Howard	2019-20	Los Angeles Lakers

Team 2

Remove All Players

View	Remove	Update	Name	Season	Team
View Stats	Remove Player	Update Player	Kawhi Leonard	2019-20	Los Angeles Clippers
View Stats	Remove Player	Update Player	Paul George	2019-20	Los Angeles Clippers
View Stats	Remove Player	Update Player	Patrick Beverley	2019-20	Los Angeles Clippers

Game Simulator:

Team 1 Win Percentage: 0.724

Team 2 Win Percentage: 0.19

Tie Percentage: 0.086

NoSQL Implementation:

A popular NoSQL database management system is Neo4j. Neo4j is popular because it allows for the database to be visualized as a graph, thus for our project we used Neo4j to help visualize the relationships between teammates of a single team. This functionality was helpful because it allowed us to see all the players that played with one another during a specific season. If any two individuals were debating between which championship team had the better overall team, this functionality would help you visualize the data and easily come up with the results.

NoSQL vs. SQL:

We decided to store all our data in a relational database because it was easy for us to query the data for the desired attributes via SQL. In order to form relationships between players and visualize them graphically, we used Neo4j. The combination of SQL and Neo4j was easy for us to use and understand because of the similar syntax between the two. NoSQL made it easier to demonstrate different relationships between nodes, in this case players, among other players. This is something that is somewhat challenging in an SQL database and requires much more computational memory to figure out, but a NoSQL implementation allows for all relations to already be given for each player and can be accessed much easier.

Challenges:

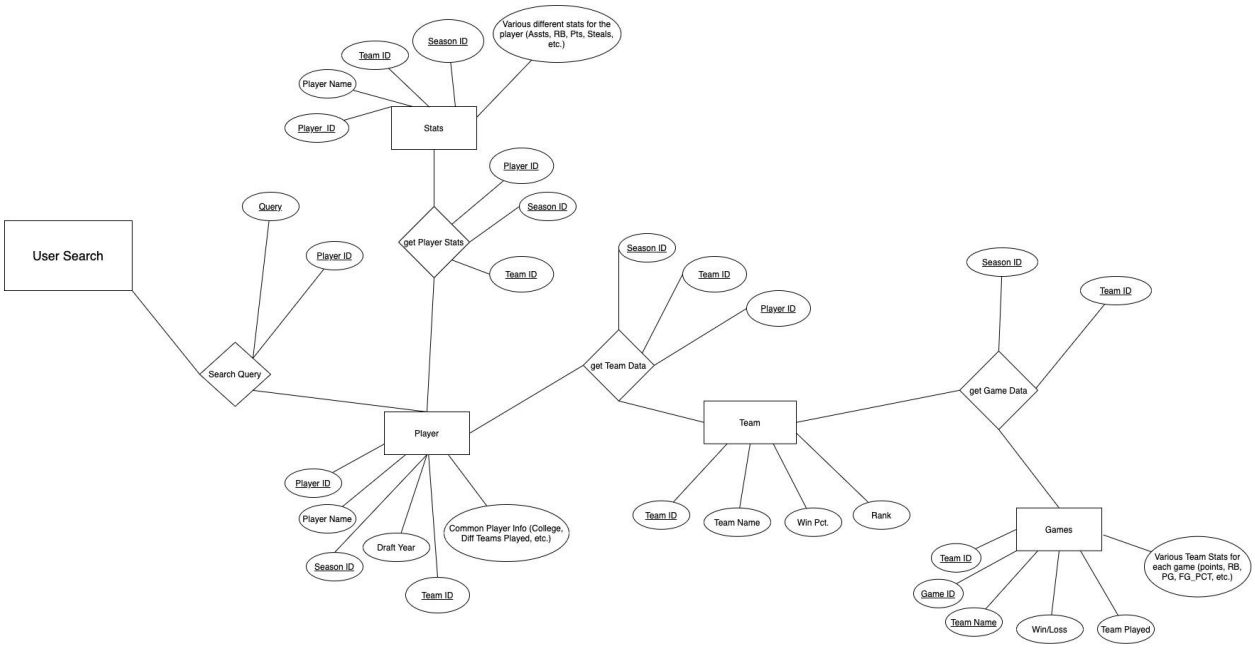
One of the biggest technical challenges that our team encountered was understanding when certain data needed to be accessed from the web application using “GET” or “POST”. During our initial attempts at this project, including our initial demo. We had the ability to create/insert records into the database as well as the ability to search for players. When first creating this application, we believed that any sort of request would allow for both types of functionalities, of receiving or posting data. However, issues started to occur when trying to implement the update and delete functionality. It first started with errors including, “This unique ID already exists in the database” and would not allow for any modifications.

We soon realized that the problem was how we were accessing the changed data as well the changes we wanted to implement to the player's teams. The biggest change that we made was in our thought process and our understanding of django. We realized that "POST" should be used whenever a user wants to modify/change the data that is given on the application. So we updated our code to check if the request being made was either a "POST," or "GET." By checking for this, we could then be certain that our code would either be accessing/outputting data to the user or if the user will be changing/modifying data that is found in the backend. Ultimately, this was beneficial for us to implement the basic CRUD functions as well as implementing the use cases we wanted to for our application.

Conclusion:

Overall, we had fun with this project because we were able to join together what we learned in class with a sport that we're interested in. Unfortunately, not all plans from our initial development stage came into fruition. Our initial plan included the ability to find players based on the best all-time stats as well as have the ability to locate individual games for a certain player of the user's choice. We were unable to properly implement those aspects into our project, however we still took the main ideas from those steps and included them in the final design of our project to provide a coherent and fun application for all NBA fans. We are happy with our project, but if we wanted to make it even better, we could make our prediction algorithm more complex in order to take more player attributes into consideration. Additionally, we could expand the functionality of our search bar to include searches for things like teams, franchises, and even injury reports. Nevertheless, we enjoyed this project and hope it can be used by basketball fans everywhere.

ER Diagram & Schema:




```
CREATE TABLE `Games` (
  `season_id` int NOT NULL,
  `team_id` int NOT NULL,
  `team_abbr` varchar(10) DEFAULT NULL,
  `team_name` varchar(45) NOT NULL,
  `game_id` int NOT NULL,
  `game_date` varchar(20) NOT NULL,
  `matchup` varchar(20) NOT NULL,
  `WL` varchar(10) DEFAULT NULL,
  `MIN` int DEFAULT NULL,
  `PTS` int DEFAULT NULL,
  `FGM` int DEFAULT NULL,
  `FGA` int DEFAULT NULL,
  `FG_PCT` float DEFAULT NULL,
  `FG3M` int DEFAULT NULL,
```

```
  `FG3A` int DEFAULT NULL,
  `FG3_PCT` float DEFAULT NULL,
  `FTM` int DEFAULT NULL,
  `FTA` int DEFAULT NULL,
  `FT_PCT` float DEFAULT NULL,
  `OREB` int DEFAULT NULL,
  `DREB` int DEFAULT NULL,
  `REB` int DEFAULT NULL,
  `AST` int DEFAULT NULL,
  `STL` int DEFAULT NULL,
  `BLK` int DEFAULT NULL,
  `TOV` int DEFAULT NULL,
  `PF` int DEFAULT NULL,
  `PlusMinus` int DEFAULT NULL,
  PRIMARY KEY (`game_id`,`team_id`,`team_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `Player` (
  `player_id` int NOT NULL,
  `season_id` varchar(10) NOT NULL,
  `team_id` int NOT NULL,
  `player_name` varchar(30) NOT NULL,
  `points` int NOT NULL,
  PRIMARY KEY (`player_id`,`season_id`,`team_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `Stats` (
```

```
`player_id` int NOT NULL,
`season_id` varchar(10) NOT NULL,
`league_id` int NOT NULL,
`team_id` int NOT NULL,
`team_abbr` varchar(10) DEFAULT NULL,
`age` int NOT NULL,
`GP` int DEFAULT NULL,
`GS` int DEFAULT NULL,
`MIN` int DEFAULT NULL,
`FGM` int DEFAULT NULL,
`FGA` int DEFAULT NULL,
`FG_PCT` float DEFAULT NULL,
`FG3M` int DEFAULT NULL,
`FG3A` int DEFAULT NULL,
`FG3_PCT` float DEFAULT NULL,
`FTM` int DEFAULT NULL,
`FTA` int DEFAULT NULL,
`FT_PCT` float DEFAULT NULL,
`OREB` int DEFAULT NULL,
`DREB` int DEFAULT NULL,
`REB` int DEFAULT NULL,
`AST` int DEFAULT NULL,
`STL` int DEFAULT NULL,
`BLK` int DEFAULT NULL,
`TOV` int DEFAULT NULL,
`PF` int DEFAULT NULL,
`PTS` int DEFAULT NULL,
```

```
  PRIMARY KEY (`player_id`,`season_id`,`team_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `Teams` (
  `id` int NOT NULL,
  `city` varchar(20) NOT NULL,
  `state` varchar(20) NOT NULL,
  `team_name` varchar(30) NOT NULL,
  `year_founded` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Code Snippets:

```
def getUserStats1(request, pk, sk):
    player_name = Players.objects.raw('SELECT * FROM Player WHERE player_id = %s', [pk])[0]
    object_list = User2Stats.objects.raw('SELECT * FROM user1stats WHERE id = %s', [sk])
    # print(object_list)
    context = {'object_list':object_list, 'player_name':player_name}
    return render(request, 'user_stats_page.html', context)

def getUserStats2(request, pk, sk):
    player_name = Players.objects.raw('SELECT * FROM Player WHERE player_id = %s', [pk])[0]
    object_list = User2Stats.objects.raw('SELECT * FROM user2stats WHERE id = %s', [sk])
    # print(object_list)
    context = {'object_list':object_list, 'player_name':player_name}
    return render(request, 'user_stats_page.html', context)

def getStats(request, pk, tk, sk):
    player_name = Players.objects.raw('SELECT * FROM Player WHERE player_id = %s AND team_id = %s AND season_id = %s', [pk, sk, tk])[0]
    object_list = Stats.objects.raw('SELECT * FROM Stats WHERE player_id = %s AND team_id = %s AND season_id = %s', [pk, sk, tk])
    # print(object_list)
    context = {'object_list':object_list, 'player_name':player_name}
    return render(request, 'stats_page.html', context)
```

```
def getSeasons(request, pk):
    object_list = Players.objects.raw('SELECT * FROM Player JOIN Teams ON Player.team_id = Teams.id WHERE player_id = %s', [pk])
    context = {'object_list':object_list}
    return render(request, 'search_results.html', context)
```

```
def best_worst_attack(request):
    teams = Teams.objects.raw('''
        (SELECT team_name, season_id, id FROM NBA.Stats JOIN NBA.Teams ON NBA.Teams.id = NBA.Stats.team_id
        GROUP BY team_id, season_id
        ORDER BY SUM(FG3_PCT) DESC, SUM(FG_PCT) DESC, SUM(AST) DESC, SUM(OREB) DESC LIMIT 5)
        UNION
        (SELECT team_name, season_id, id FROM NBA.Stats JOIN NBA.Teams ON NBA.Teams.id = NBA.Stats.team_id
        GROUP BY team_id, season_id
        ORDER BY SUM(FG3_PCT) ASC, SUM(FG_PCT) ASC, SUM(AST) ASC, SUM(OREB) ASC LIMIT 5)
                                ''')
    offense = 'Offense'
    context = {'object_list':teams, 'offense':offense}
    return render(request, 'complex2.html', context)

def best_worst_defense(request):
    teams = Teams.objects.raw('''
        (SELECT team_name, season_id, id FROM NBA.Stats JOIN NBA.Teams ON NBA.Teams.id = NBA.Stats.team_id
        GROUP BY team_id, season_id
        ORDER BY SUM(DREB) DESC, SUM(STL) DESC, SUM(BLK) DESC LIMIT 5)
        UNION
        (SELECT team_name, season_id, id FROM NBA.Stats JOIN NBA.Teams ON NBA.Teams.id = NBA.Stats.team_id
        GROUP BY team_id, season_id
        ORDER BY SUM(DREB) ASC, SUM(STL) ASC, SUM(BLK) ASC LIMIT 5)
                                ''')
    defense = 'Defense'
    context = {'object_list':teams, 'defense':defense}
    return render(request, 'complex2.html', context)
```

```
def viewLevel(request, pk, sk):
    lower = int(pk) - 250
    higher = int(pk) + 250
    print(lower, higher)
    players = Players.objects.raw('''
    SELECT * FROM NBA.Player JOIN NBA.Teams ON NBA.Player.team_id = NBA.Teams.id
WHERE points > %s AND points < %s AND (team_id, season_id) IN (SELECT team_id, season_id FROM NBA.Player WHERE season_id = %s)
ORDER BY points ASC
''', [lower, higher, sk])
    context = {'object_list':players}
    return render(request, 'search_results.html', context)
```

```python
class Neo4jConnection:

    def __init__(self, uri, user, pwd):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__driver = None
        try:
            self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
        except Exception as e:
            print("Failed to create the driver:", e)

    def close(self):
        if self.__driver is not None:
            self.__driver.close()

    def query(self, query, db=None):
        assert self.__driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.__driver.session(database=db) if db is not None else self.__driver.session()
            response = list(session.run(query))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response
```

```
def viewTeammates(request, pk, sk, tk):
    # players = Players.objects.all()
    print('YO')
    conn = Neo4jConnection(uri="bolt://localhost:7687", user="neo4j", pwd="password123")
    print('YO')
    s = '''
                    MATCH (a:Player)-[r:TEAMMATES_WITH]->(b:Player)
                    WHERE a.playerID = {0} AND a.seasonID = '{1}'
                    RETURN b
                    '''
    query_string = s.format(pk, sk)
    print(query_string)
    list = []
    result = conn.query(query_string)
    for node in result:
        list.append(node['b']['playerName'])
        # print('YO')
    in_list = '('
    for i, player in enumerate(list):
        if i == (len(list) - 1):
            in_list += str(player)
        else:
            in_list += str(player)
            in_list += ', '
    in_list += ')'
    object_list = Players.objects.filter(player_name__in=list, season_id=sk, team_id=tk)
    team_name = Teams.objects.raw('SELECT * FROM Teams WHERE id = %s', [tk])[0]
    print(getattr(team_name, 'team_name'))
    context = {'object_list':object_list, 'team_name':team_name}
    return render(request, 'team_mates.html', context)
```