Bilkent University

Department of Computer Engineering

CS 353 – Database Systems

Project Design

Kumpir: Football Database System

Spring 2018

Group 11:

Berat BİÇER, 21503050

Ecem İLGÜN, 21502157

Ahmet Batu ORHAN, 21402365

Serdar TAŞKAFA, 21501927

Teaching Assistant: Arif Usta

Bilkent University

March 29, 2018

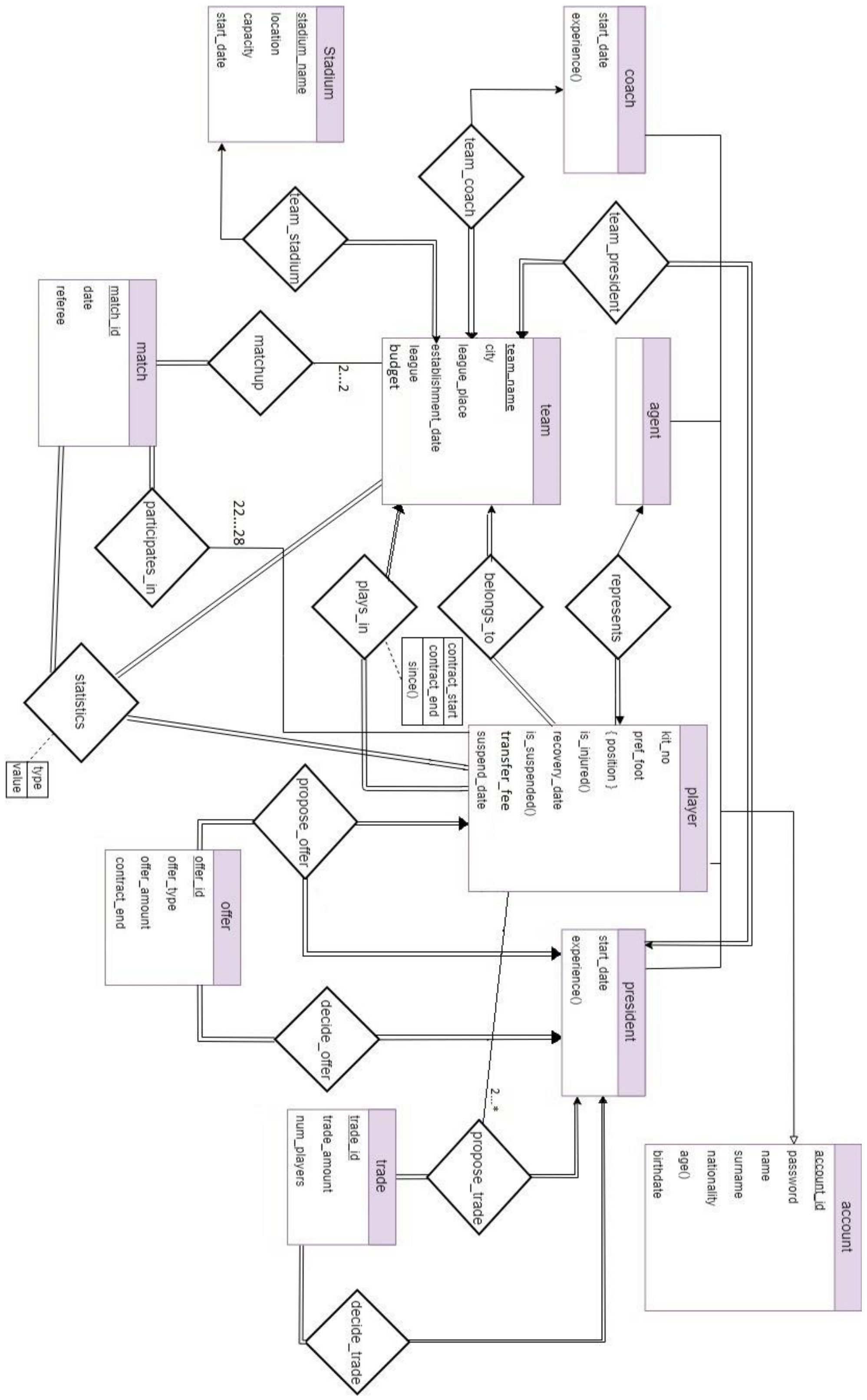
TABLE OF CONTENTS

1. REVISED E/R MODEL

We have made the following changes to our initial E/R design after proposal feedback and during design report's discussions.

- We have removed **home** and **guest** relationships, and have merged them into a single relationship called **matchup** instead. **home** relationship was 0 to 1 on both of its **Team** and **Match** sides, the same held for **guest** as well. After the merge, we have enforced total participation as well and now **matchup** relationship enforces exactly 2 **Team** entities and at least 1 **Match** between them.
- We used to hold statistics as 6 different attributes(player.stat_exp, player.player_stats, team.team_stats, team.stat_exp, match.match_stats, match.stat_exp) which we used to distribute among **Team**, **Match** and **Player** entities. We have first thought to encode statistic explanations in stat_exp attributes and statistics themselves in match_stats, player_stats and team_stats attributes. We switched our previous approach of encoding information in application program to storing it in database. With that purpose, we have created a **Statistics** entity instead. **Statistics** entity holds a primary key of player.account_id, match.match_id and team.team_name. It also holds type and value attributes. Type attribute can be used to represent any type of statistics (goal, assist etc.) and value will be an integer (since statistics are bound to be numbers).
- We have replaced the primary key of team_id on **Team** entity with team_name instead, since we believe that each **Team** should hold a unique name, which deemed team_id unnecessary. We have added establishment_date and league place attributes to provide further information. We have deleted league attribute since there wasn't any league structure within our system, which made this attribute unnecessary. We now have built a system which focuses on only one league. We have removed team_stats and stat_exp attributes, due to design choices which we have explained in **Statistics** change above.
- We don't hold a list of previous teams of **Player** entity in prev_team attribute anymore. We changed the database to more revolve around **Statistics** so we didn't need that information anymore. We have added a new attribute under the name of prev_transfer_fee, which holds the latest transfer fee of the **Player**, in order to estimate his value. We have removed player_stats and stat_exp attributes, due to design choices which we have explained in **Statistics** change above.
- We have removed match_stats and stat_exp attributes from **Match**, due to design choices which we have explained in **Statistics** change above.
- In our previous iteration, we had an **Person**(<u>person_id</u>, name, surname, nationality, birthdate) entity which was in a 1-1 relationship with **Account**(<u>username</u>, password) entity. We have merged them in to single **Account**(<u>account_id</u>, password, name, surname, nationality, birthdate) entity.
- We have enforced total participation on **decides_trade** relationship on both **President** and **Trade** sides. We have also changed cardinality of **President** side of the relationship to exactly 1.
- We have enforced total participation on **propose_trade** relationship on both **President, Player** and **Trade** sides. We have changed cardinality of **propose_trade** relationship's **President** side to exactly 1, and **Player** side to at least 2.
- We have removed time limit from **Trade** entity. We now use contract ending times only for rental processes, trade and transfers are decided to be for indefinite time.

- We have enforced total participation on **decides_offer** relationship on both **President** and **Offer** sides. We have also changed cardinality of **President** side of the relationship to exactly 1.
- We have enforced total participation on **propose_offer** relationship on both **President, Player** and **Offer** sides. We have changed cardinality of **propose_trade** relationship's **President** and **Player** side to exactly 1.
- We have enforced total participation on **represent** relationship's Player side. With that we eliminated free agents.
- We have enforced total participation on **belongs_to** relationship on both **Player** and **Team** sides. With that we eliminated players who did not belong to a team.
- We have enforced total participation on **plays_in** relationship on both **Player** and **Team** sides. Other than that, we have added 3 attributes to **plays_in** relationship: contract_start, contract_end and a derived attribute since(). This helps us with the cases where **Player** is rented to another team for a limited time and is expected to return back to the team he belongs to.
- We have changed time_limit attributes name to contract_end in **Offer** entity for consistency.
- We have enforced total participation on both **Match** and **Player** sides of **participates_in** relationship. Other than that, we have restricted cardinality of Player side to be between 22 and 28.
- We have enforced total participation on both **Team** and **President** sides of **team_president** relationship. Their cardinality used to be 0 to 1, now it is strictly 1.
- We have enforced total participation on **Team** side of **team_coach** relationship. Now we allow a team to not have a coach, but each coach are strictly restricted to one and only one team.
- We have enforced total participation on **Team** side of **team_stadium** relationship. Now we allow a team to not have a stadium, but each stadium are strictly restricted to one and only one team.


Stadium
stadium_name
location
capacity
start_date
coach
start_date
experience()
team_coach
team_stadium
team_president
match
match_id
date
referee
matchup
2...2
team
team_name
city
league_place
establishment_date
league
budget
agent
participates_in
22...28
plays_in
belongs_to
represents
contract_start
contract_end
since()
statistics
type
value
player
kit_no
pref_foot
{ position }
is_injured()
recovery_date
is_suspended()
transfer_fee
suspend_date
propose_offer
offer
offer_id
offer_type
offer_amount
contract_end
president
start_date
experience()
decide_offer
2...*
trade
trade_id
trade_amount
num_players
propose_trade
account
account_id
password
name
surname
nationality
age()
birthdate
decide_trade

2. RELATION SCHEMAS

2.1. PLAYER

Relational Model:

player(<u>account_id</u>, password, name, surname, nationality, birthdate, kit_no, pref_foot, recovery_date, suspend_date, belong_to_team_name, prev_transfer_fee)

Functional Dependencies:

account_id -> account_id, password, name, surname, nationality, birthdate, kit_no, pref_foot, recovery_date, suspend_date, belong_to_team_name, prev_transfer_fee

Candidate Key:

{account_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE player (

account_id varchar(255) primary key,

password varchar(255) not null,

name varchar(255) not null,

surname varchar(255) not null,

nationality varchar(255),

birthdate varchar(255),

kit_no int,

pref_foot int,

prev_transfer_fee int not null,

recovery_date varchar(255),

suspend_date varchar(255),
```

belong_to_team_name varchar(255) not null,

foreign key(belong_to_team_name) references to team(team_name));

2.2. POSITION

Relational Model:

position(account_id, position)

Functional Dependencies:

account_id, position -> account_id, position

Candidate Key:

{account_id, position}

Normal Form:

BCNF

Table Definition:

```
create table position (
account_id varchar(255) ,
position varchar(255) ,
primary key(account_id, position),
foreign key(account_id) references player(account_id)
);
```

2.3. PRESIDENT

Relational Model:

president(account_id, password, name, surname, nationality, birthdate, start_date, team_name)

Functional Dependencies:

account_id -> account_id, password, name, surname, nationality, birthdate, start_date, team_name

Candidate Key:

{account_id}

Normal Form:

BCNF

Table Definition:

create table president (

account_id varchar(255) primary key,

password varchar(255) not null,

name varchar(255) not null,

surname varchar(255) not null,

nationality varchar(255),

birthdate varchar(255),

start_date varchar(255),

team_name varchar(255) not null,

foreign key(team_name) referencing team(team_name)

);

2.4. COACH

Relational Model:

coach(account_id, password, name, surname, nationality, birthdate, start_date, team_name)

Functional Dependencies:

account_id -> account_id, password, name, surname, nationality, birthdate, start_date, team_name

Candidate Key:

{account_id}

Normal Form:

BCNF

Table Definition:

```
create table coach (

account_id varchar(255) primary key,

password varchar(255) not null,

name varchar(255) not null,

surname varchar(255) not null,

nationality varchar(255),

birthdate varchar(255),

start_date varchar(255),

team_name varchar(255) not null,

foreign key(team_name) references team(team_name)

);
```

2.5. AGENT

Relational Model:

agent(account_id, password, name, surname, nationality, birthdate, player_account_id)

Functional Dependencies:

account_id -> account_id, password, name, surname, nationality, birthdate, player_account_id

Candidate Key:

{account_id}

Normal Form:

BCNF

Table Definition:

```
create table agent(

account_id varchar(255) primary key,

password varchar(255) not null,

name varchar(255) not null,

surname varchar(255) not null,

nationality varchar(255),

birthdate varchar(255),

player_account_id varchar(255) not null,

foreign_key(player_account_id) references player(account_id)

);
```

2.6. TEAM

Relational Model:

team(<u>team_name</u>, city, league_place, stadium_name, budget, establishment_date)

Functional Dependencies:

team_name -> team_name, city, league_place, stadium_name, budget, establishment_date

Candidate Key:

{team_name}

Normal Form:

BCNF

Table Definition:

```
create table team (

team_name varchar(255) primary key,

city varchar(255) not null,

league_place varchar(255) not null,

stadium_name varchar(255) not null,

budget int not null,

establishment_date date,

foreign key(stadium_name) references stadium(stadium_name),

);
```

2.7. MATCH

Relational Model:

match(match_id, date, referee, home_team_name, guest_team_name)

Functional Dependencies:

match_id -> match_id, date, referee, home_team_name, guest_team_name

Candidate Key:

{match_id}

Normal Form:

BCNF

Table Definition:

```
create table match (
match_id varchar(255) primary key,
date varchar(255),
referee varchar(255),
home_team_name varchar(255) not null,
guest_team_name varchar(255) not null,
foreign key(home_team_name) references team(team_name),
foreign key(guest_team_name) references team(team_name),
);
```

2.8. STADIUM

Relational Model:

stadium(<u>stadium_name</u>, location, capacity, start_date)

Functional Dependencies:

stadium_name -> stadium_name, location, capacity, start_date

Candidate Key:

{stadium_name}

Normal Form:

BCNF

Table Definition:

```
create table stadium (
stadium_name varchar(255) primary key,
location varchar(255),
capacity INT,
start_date varchar(255)
);
```

2.9. PLAYS_IN

Relational Model:
plays_in(<u>team_name, account_id</u>, contract_start, contract_end)

Functional Dependencies:
team_name, account_id -> team_name, account_id, contract_start, contract_end

Candidate Keys:
{team_name, account_id}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE plays_in (
        team_name varchar(255) not null,
        account_id varchar(255) not null,
        contract_start int,
        contract_end int not null,
        PRIMARY KEY(team_name, account_id),
        FOREIGN KEY (team_name) references team(team_name),
        FOREIGN KEY (account_id) references player(account_id)
);
```

2.10. OFFER

Relational Model:
offer(<u>offer_id</u>, offer_type, offer_amount, contract_end, decidepresident_account_id)

Functional Dependencies:
offer_id -> offer_id, offer_type, offer_amount, contract_end, decidepresident_account_id

Candidate Keys:
{offer_id}

Normal Form:
BCNF

Table Definition:
```
create table offer(
        offer_id varchar(255) primary key,
        offer_type varchar(255) not null,
        offer_amount int not null,
        contract_end int not null,
        decidepresident_account_id varchar(255) not null,
        foreign key(decidepresident_account_id) references president(account_id)
);
```

2.11. TRADE

Relational Model:

trade(<u>trade_id</u>, trade_amount, time_limit, num_players, decidepresident_account_id)

Functional Dependencies:

trade_id -> trade_id, trade_amount, time_limit, num_players, decidepresident_account_id

Candidate Keys:

{trade_id}

Normal Form:

BCNF

Table Definition:

```
create table trade (
        offer_id varchar(255) primary key,
        offer_type varchar(255) not null,
        offer_amount int not null,
        decidepresident_account_id varchar(255) not null,
        foreign key(decidepresident_account_id) references president(account_id)
);
```

2.12. PROPOSE_OFFER

Relational Model:
propose_offer{<u>offer_id, player_account_id, president_account_id</u>}

Functional Dependencies:
offer_id, player_account_id, president_account_id -> offer_id, player_account_id, president.president_account_id

Candidate Keys:
{offer_id, player_account_id, president_account_id }

Normal Form:
BCNF

Table Definition:
```
create table propose_offer (
        offer_id varchar(255),
        player_account_id varchar(255),
        president_account_id  varchar(255),
        primary key (offer_id, player_account_id, president_player_id),
        foreign key (offer_id) references offer(offer_id),
        foreign key (player_account_id) references player(account_id),
        foreign key (president_account_id) references president(account_id)
);
```

2.13. PROPOSE_TRADE

Relational Model:
propose_trade{<u>trade_id, player_account_id, president_account_id</u>}

Functional Dependencies:
trade_id, player_account_id, president_player_id -> trade_id, player_account_id, president.player_id

Candidate Keys:
{trade_id, player_account_id, president_player_id}

Normal Form:
BCNF

Table Definition:
```
create table propose_trade (
        trade_id varchar(255),
        player_account_id varchar(255),
        president_player_id varchar(255),
        primary key (trade_id, player_account_id, president_player_id),
        foreign key (trade_id) references trade(trade_id),
        foreign key (player_account_id) references player(account_id),
        foreign key (president_account_id) references president(account_id)
);
```

2.14. PARTICIPATES_IN

Relational Model:
participates_in(<u>account_id, match_id</u>)

Functional Dependencies:
account_id, match_id -> account_id, match_id

Candidate Keys:
{account_id, match_id}

Normal Form:
BCNF

Table Definition:
```
create table participates_in (
        account_id varchar(255),
        match_id varchar(255),
        primary key(account_id, match_id)
);
```

2.15. STATISTICS

Relational Model:
statistics(match_id, team_name, account_id, type, value)

Functional Dependencies:
match_id, team_name, account_id, type -> match_id, team_name, account_id, type, value

Candidate Keys:
{match_id, team_name, account_id, type}

Normal Form:
BCNF

Table Definition:
```
create table statistics (
        match_id varchar(255),
        team_name varchar(255),
        account_id varchar(255),
        type varchar(255),
        value int not null,
        primary key(match_id, team_name, account_id, type),
        foreign key(match_id) references match(match_id),
        foreign key(team_name) references team(team_name),
        foreign key(account_id) references player(account_id)
);
```

3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

All tables in our database is in normal form; therefore, there is no normalization required.

4. FUNCTIONAL COMPONENTS

4.1. USE CASES / SCENARIOS

4.1.1. ADMINISTRATOR

Create account: Administrator can create a new account, insert it to the database, and set connections between it and others. The term account involves players, coaches, presidents and agents.

Edit Account: Administrator can change the status of an account, its owner, corresponding data, and its connections to other entities. The changes are reflected on the database.

Delete Accounts: Administrator can delete an existing user account by first finding that account, then deleting it from the database, and removing its connections.

Display Account: Administrator can display existing accounts in the system by a search on database. This use case is publicly accessible; however, results are filtered in this case.

Display Search Result Details: After executing a search administrator may select one of the search results and display its details. The details depend on result: an account will display information of that account whereas a unit will display information of that unit. Details are retrieved from the database based on whether attributes are private. For example, a public user may not see the password of a player’s account. This use case is publicly accessible; however, results are filtered in this case.

Search: Administrator can enter the website, execute searches using keywords, and display search results. Results are taken from the database and filtered for simplicity depending on relevance.

Create Offer: Administrator can create new accounts, insert it to the database, and set connections between it and others. The term offer includes trade and transfer offers.

Edit Offer: Administrator can edit offers by first finding the offer with a search on database.

Delete Offer: Administrator can delete offers by finding the offer with a search, then deleting it from the database with a query. Lastly, he will update the actors involved if necessary.

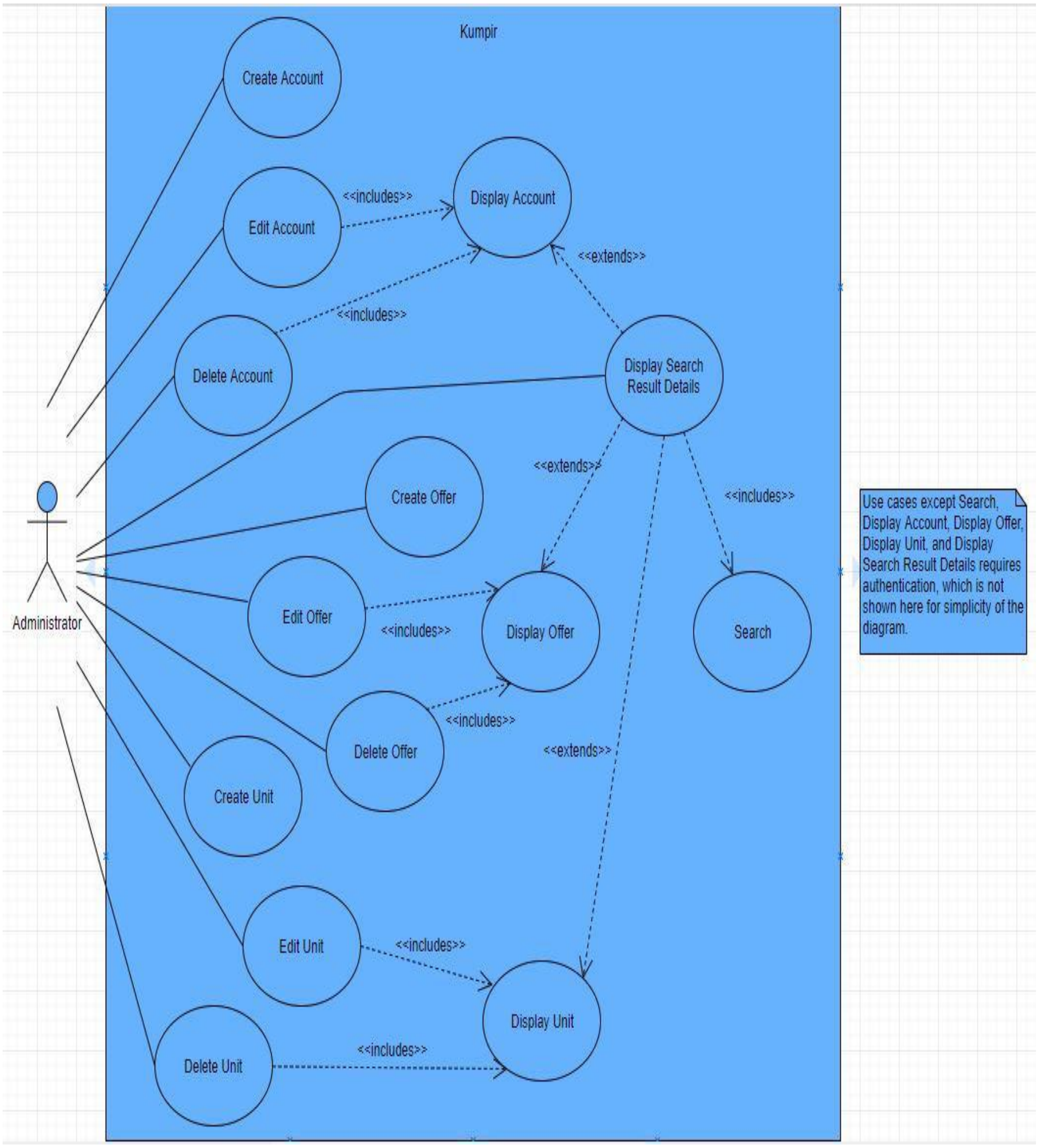
Display Offer: Administrator can see all the details on database related to a specific offer, with the option to filter the result. To do so, a search query is executed on the database. This use case is publicly accessible; however, results are filtered in this case.

Create Unit: Administrator can create new accounts, insert it to the database, and set connections between it and others. The term unit includes team, match, stadium, and relation tables.

Edit Unit: Administrator can edit units by first finding it with a search query on database.

Delete Unit: Administrator can delete units by finding the unit with a search, then deleting it from the database with a query. Lastly, he will update the actors involved if necessary.

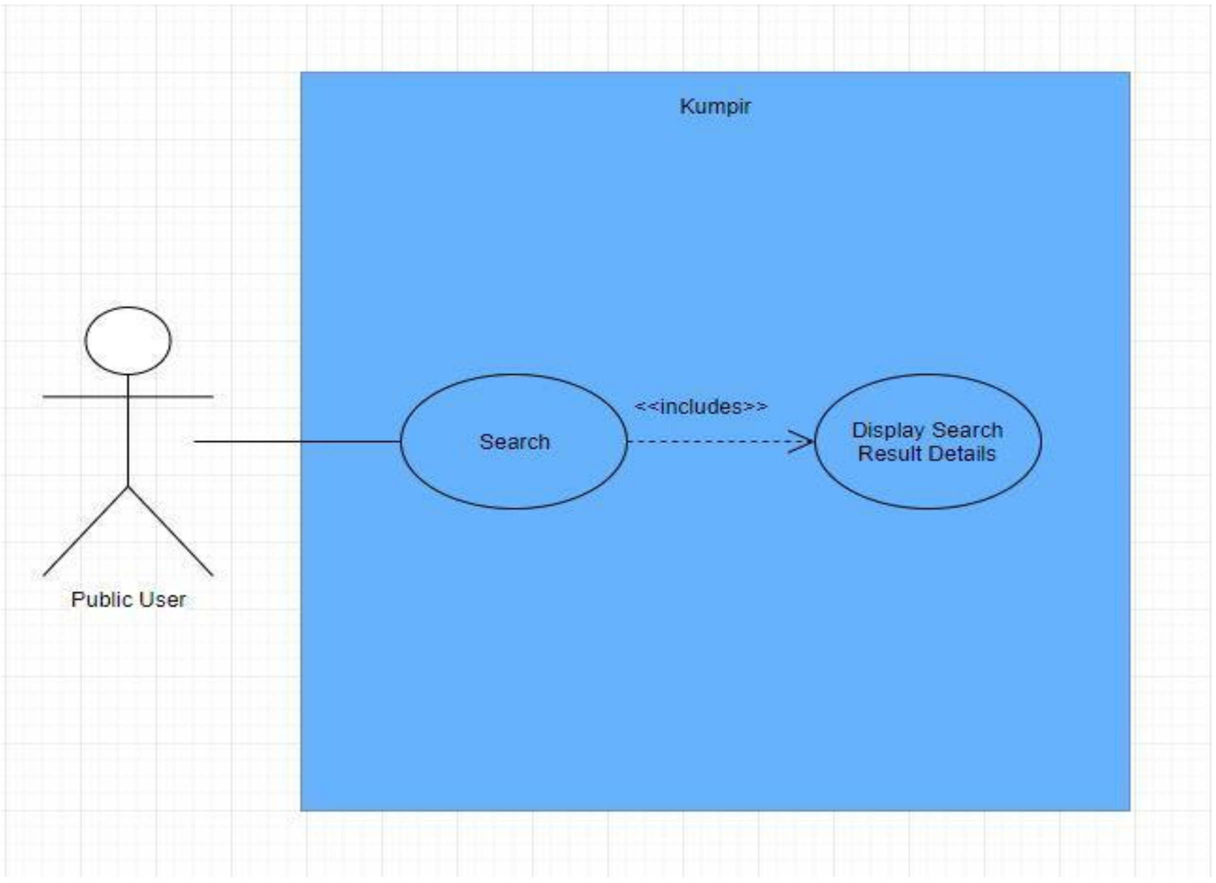
Display Unit: Administrator can see all the details on database related to a specific unit, with the option to filter the result. To do so, a search query is executed on the database. This use case is publicly accessible; however, results are filtered in this case.



4.1.2. PUBLIC USER

Search: Public user can enter the website, execute searches using keywords, and display search results.

Display Search Result Details: After executing a search public user may select one of the search results and display the details of the result. The details depend on the type of the result selected: for example, a selected entry of a player will display information of that play stored in the database whereas an entry of a team will display information related to that team.



4.1.3. PLAYER

Search: Player can enter the website, execute searches using keywords, and display search results.

Display Search Result Details: After executing a search a player may select one of the search results and display the details of the result. The details depend on the type of the result

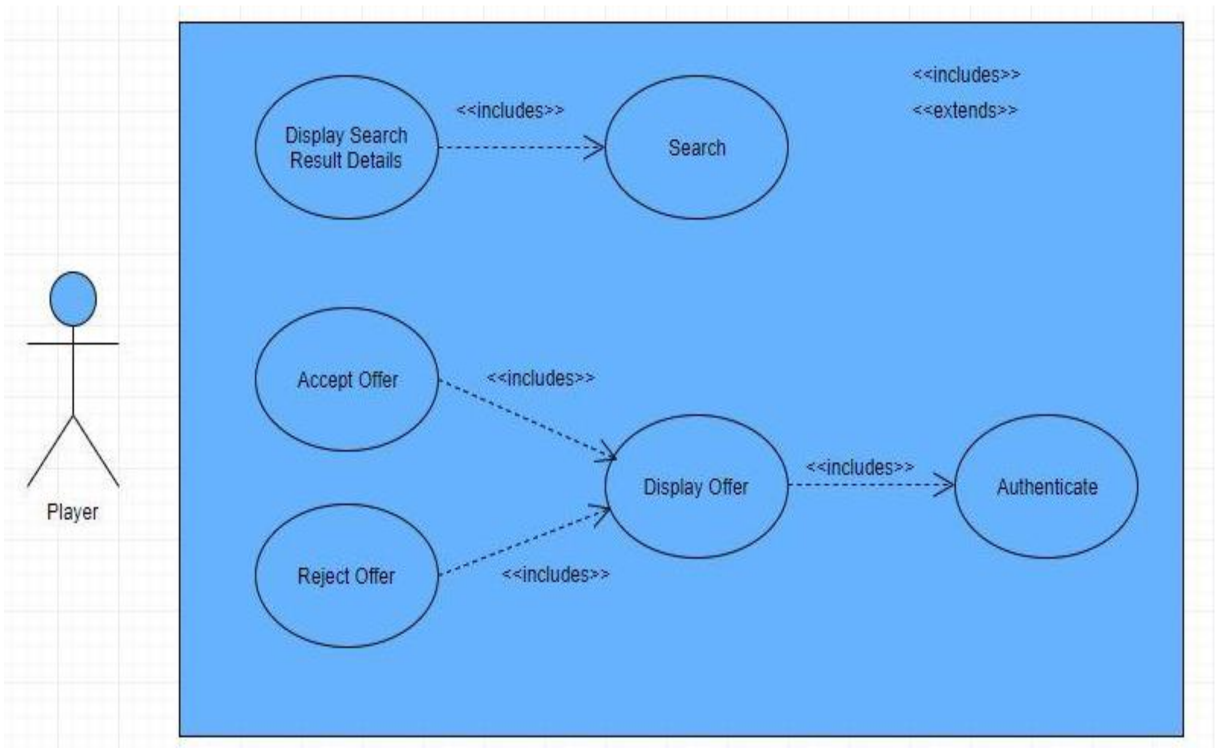
selected: for example, a selected entry of a player will display information of that play stored in the database whereas an entry of a team will display information related to that team.

Authenticate: Player needs to authenticate before accessing other use cases except search and display search results.

Display Offer: Player is able to display offers to him, made by a president.

Accept Offer: Player can accept an offer, if he has any.

Refuse Offer: Player can reject an offer, if he has any.



4.1.4. AGENT

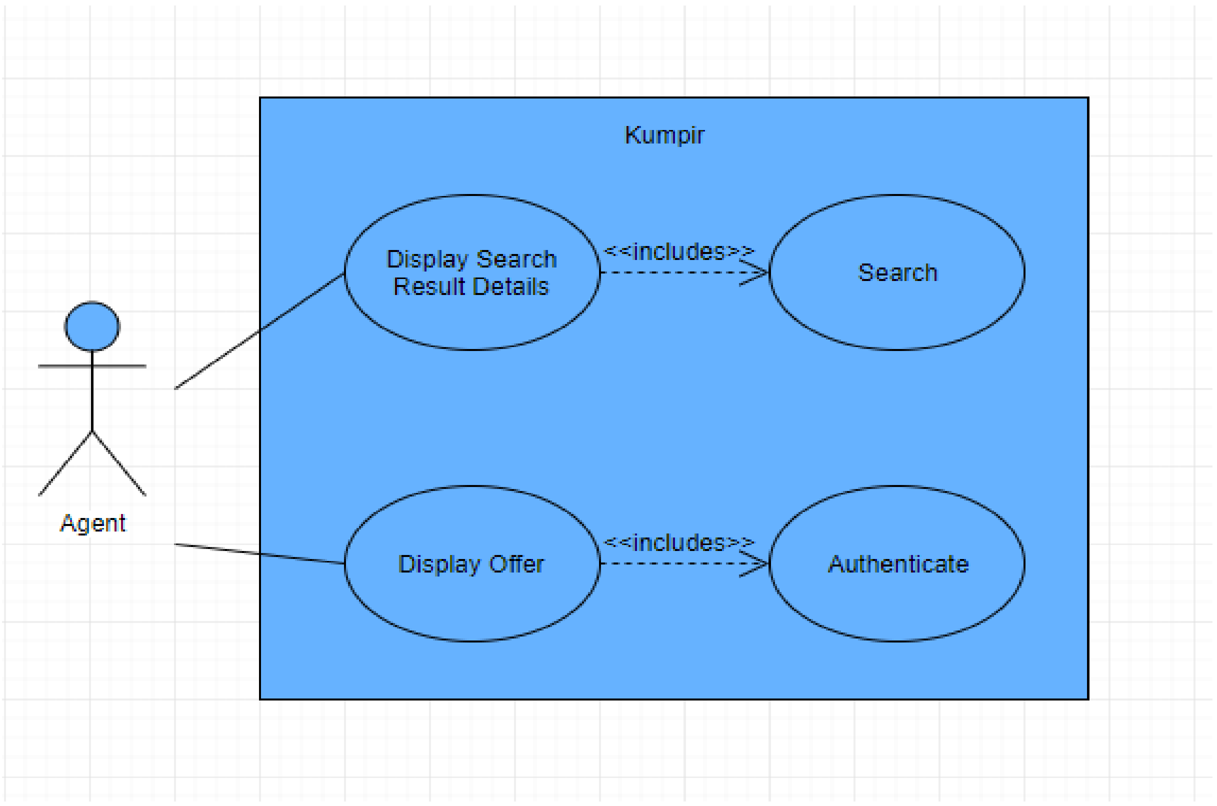
Search: An agent can enter the website, execute searches using keywords, and display search results.

Display Search Result Details: After executing a search public user may select one of the search results and display the details of the result. The details depend on the type of the result

selected: for example, a selected entry of a player will display information of that play stored in the database whereas an entry of a team will display information related to that team.

Authenticate: Player needs to authenticate before accessing other use cases except search and display search results.

Display Offer: Player is able to display offers to him, made by a president.



4.1.5. PRESIDENT

Search: President can enter the website, execute searches using keywords, and display search results.

Display Search Result Details: After executing a search a president may select one of the search results and display the details of the result. The details depend on the type of the result

selected: for example, a selected entry of a player will display information of that play stored in the database whereas an entry of a team will display information related to that team.

Authenticate: President needs to authenticate before accessing other use cases except search and display search results.

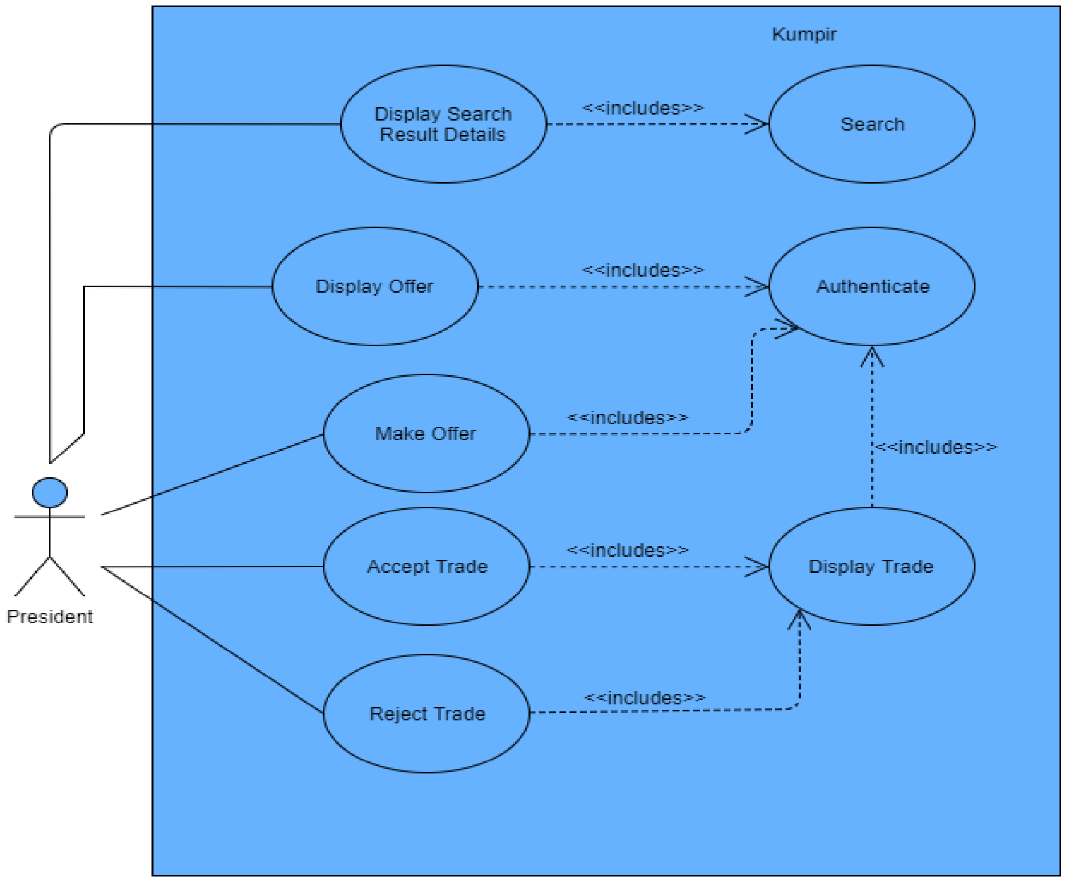
Display Offer: President is able to display offers to one of the players belongs to the team, made by another team's president.

Make Offer: President is able to make offers to the other teams' players.

Display Trade: President can display trade offers that made by another team's president.

Accept Trade: Presidents can accept trade offers, if there are any.

Refuse Trade: Presidents can refuse trade offers, if there are any.



4.1.6. COACH

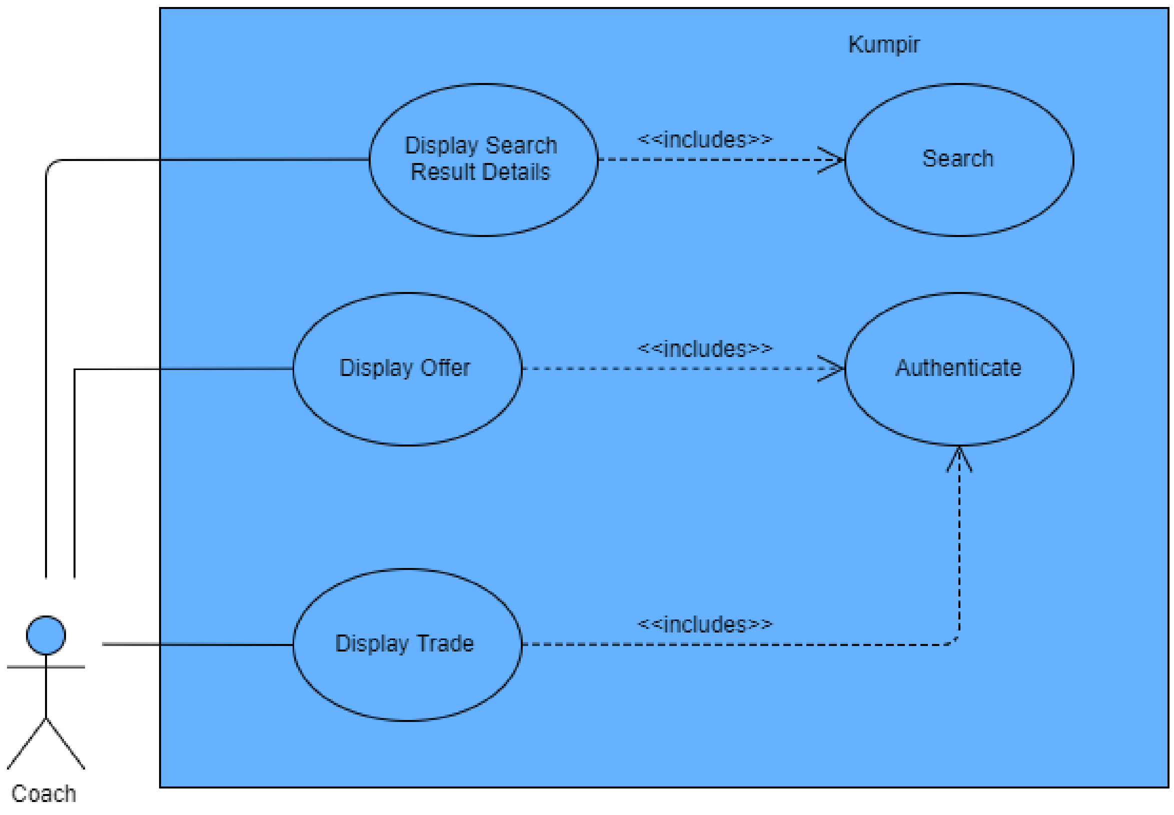
Search: President can enter the website, execute searches using keywords, and display search results.

Display Search Result Details: After executing a search a president may select one of the search results and display the details of the result. The details depend on the type of the result selected: for example, a selected entry of a player will display information of that play stored in the database whereas an entry of a team will display information related to that team.

Authenticate: President needs to authenticate before accessing other use cases except search and display search results.

Display Offer: President is able to display offers to one of the players belongs to the team, made by another team's president.

Display Trade: President can display trade offers that made by another team's president.



4.2. ALGORITHMS

4.2.1. LOGIN AND AUTHENTICATION

When the user goes to login screen, they are asked to provide an username, a password, and a user type. User type is used to determine the account type such as a player or an agent. When a user creates an account, username creator generates a username for that account using its account id. Similar to account id, password too is generated by the system for security reasons and given to the user.

If the input is in invalid format such as empty strings, access is denied. Else, a search query is executed in the database targeting the corresponding user type, which is also an input. However, account type can only be selected among a set of predetermined values, therefore there is no need to validate account type. If the search query returns a non-empty tuple from the database, the user is granted access to the system; allowing them access to privileged data and functions such as viewing offers on a player or creating new offers.

4.2.2. OFFERS

4.2.2.1. TRANSFER / RENT REQUEST

To make a transfer offer, a president first searches for a player in the database by executing a search query. There, he may select a player of his liking and create a transfer offer using that player. Insertions on propose_offer and decide_offer tables are made after president fills request details properly. These include offer type (as transfer or renting offer), offer amount and duration of the contract. After the request is created, it is sent to the player on focus and president of the team that player currently belongs to. If both of them confirms the request, the transfer occurs and following queries are executed by the system:

- Update query on the team the player belongs to and plays in, depending on offer type.
- Update query on budget of the offer-making and offer-receiving teams

Following constraints are verified before the transfer offer is created:

- Budget of the offer-making team should be more than the transfer cost.
- Player involved in an offer couldn't have been transferred in the past 6 months.
- A president cannot make an offer to players who play in his own team.
- If a player is rented, a transfer offer cannot be made to that player.

4.2.2.2. TRADE REQUEST

A president may create a trade offer by first selecting players from his team and exactly one other team. Then, he may choose an amount to be paid to or received from the other team for the trade. Then, a trade request is created and insertions to propose_trade and decide_trade tables are made. Then, trade request is sent to players involved in the request and the president of the other team. If all players and the president confirms the request, a trade occurs and following update queries are executed on the database:

- Update query on the budget of the offer-making team and/or offer-receiving team, if there is an amount in the trade offer.
- Update query to change team informations of the players who are subject to the trade offer.

Following constraints are verified before a trade offer is created:

- An offer must include at least 2 players, 1 from offer-making president's team and 1 from offer-receiving president's team.
- If a player is rented, he cannot be included in trade offers.
- There must be exactly 2 teams included in the offer. In other words, there must be exactly 2 distinct presidents involved.
- A player who is recently transferred, traded or rented is suspended and therefore cannot be included in trades.

4.2.3. STATISTICS AND MATCH-UP

Admin is the only user in the system who is able to create match and statistics entries. Upon selecting two distinct team, admin executes a query to insert a tuple into match table. This tuple contains the home and guest teams. Following constraints are satisfied before creating the tuple:

- There must be at least 2 matches among each pair of teams in each league season.
- Time between consecutive matches between the same pair must be greater than or equal to 1 month

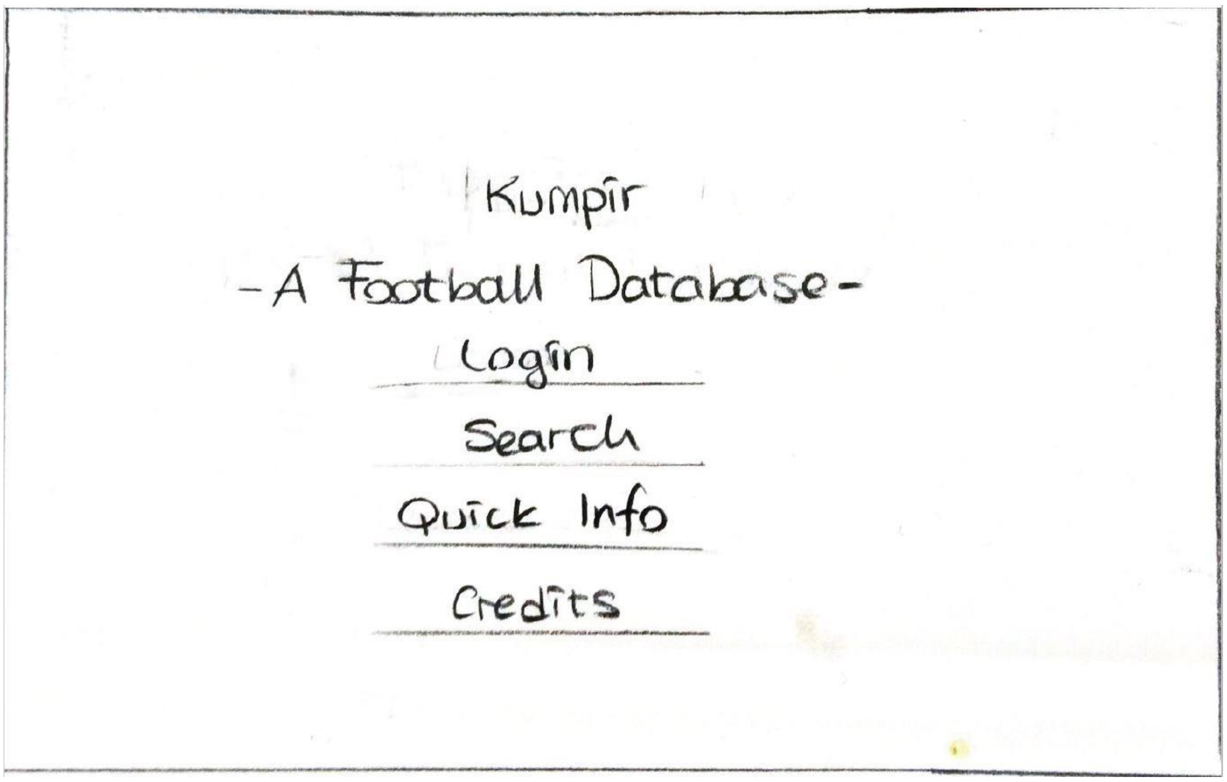
- If the capacity of the home team's stadium is less than 10.000 people, a new stadium must be selected for the match.

After a match is created, admin executes a query to insert a tuple into statistics table for each player participated in the match. Following constraints hold for statistics table:

- Each statistics tuple must be created by the admin within 3 days of the match date
- Number of distinct type of statistics for each player in each match is fixed. For example, for each player in a match there must be exactly N distinct statistics type as goal, assist, etc. where N is a predetermined number.

5. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

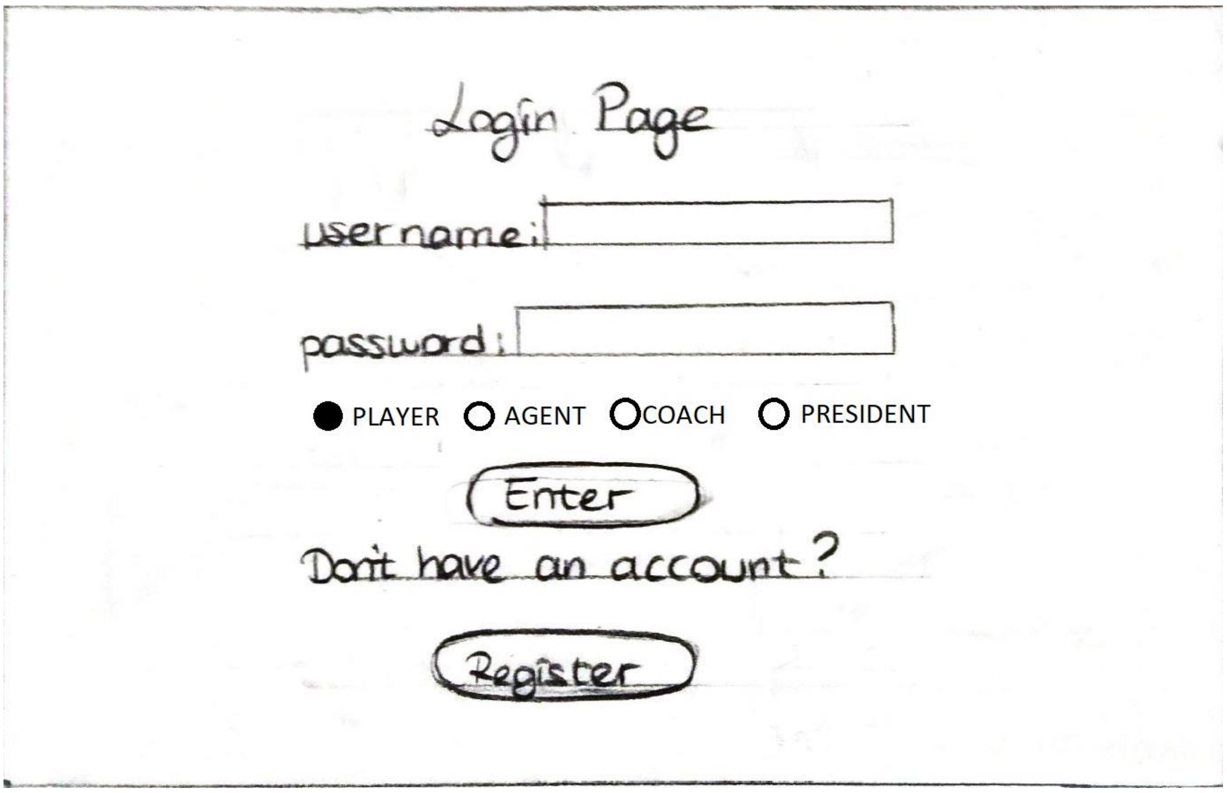
5.1. MAIN PAGE



Inputs: None

Process: Homepage is the default page of the website. Whenever a user enters the URL of the project, this page is shown. User then is able to choose the following options: Login to access privileged functionalities such as making a transfer offer by pressing Login button, search for entries in database by clicking to Search button, see some selected statistics for public display from the database by pressing Quick Info button, display credentials of our group members by pressing Credits button. This page has no direct SQL queries to be executed, it serves as a linker.

5.2. LOGIN PAGE



Inputs: @username, @password, @user_type

Process: Upon pressing Login button on the main page, the user is directed here. Then, user is asked to enter a valid username corresponding password, and selecting the account type. Then, if the data matches with an entry, the user is granted access. The user may also choose to not enter a username but to click the register button. In that case, login page acts as a linker to the registration page.

@username: User name of the client, with which he can authorize
@password: Account password of the user, used for authentication
@user_type: Determines which table the search query will be executed on

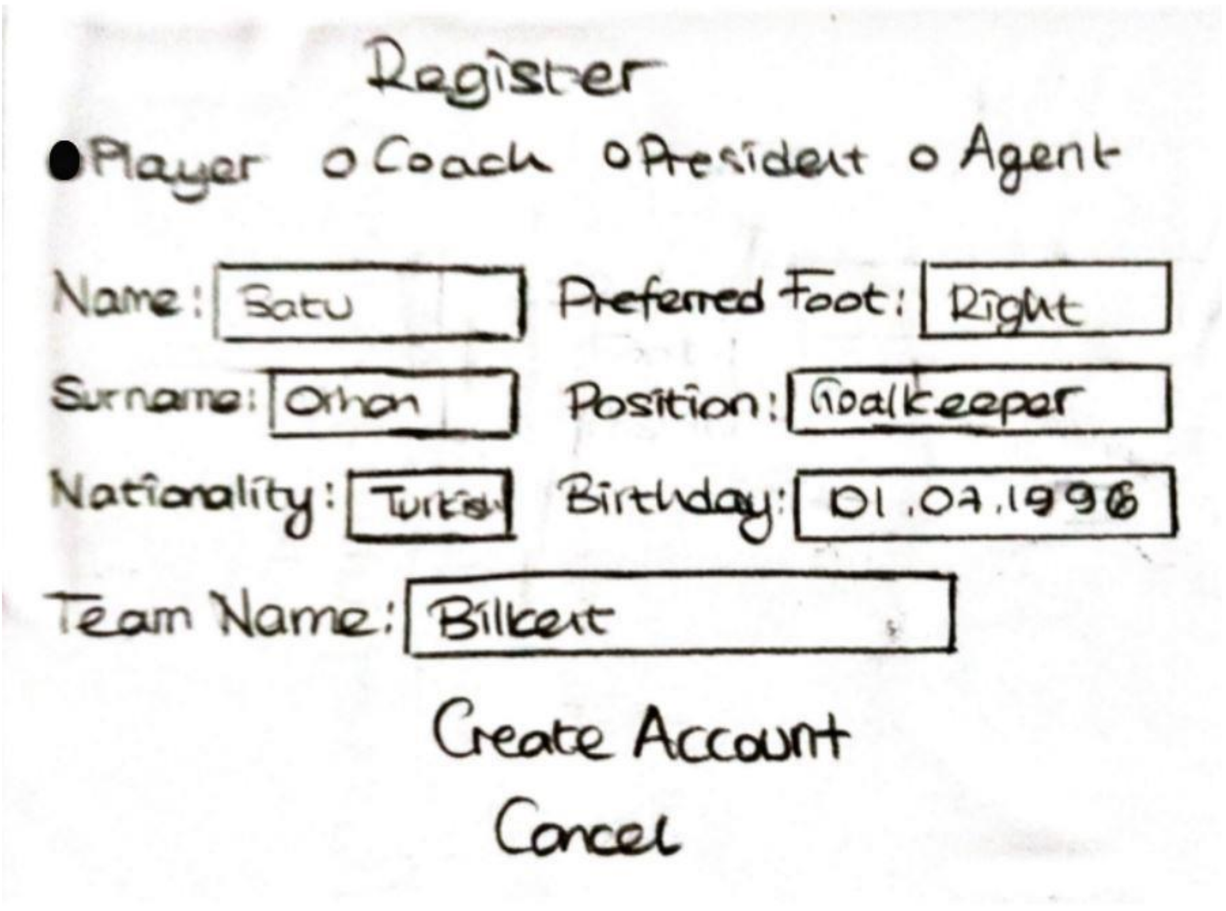
SQL Statements:

Searching DB for an account with given type, username and password:

SELECT account_id, password

```
FROM @user_type
WHERE account_id = @account_id AND password = @password;
```

5.3. REGISTRATION PAGE



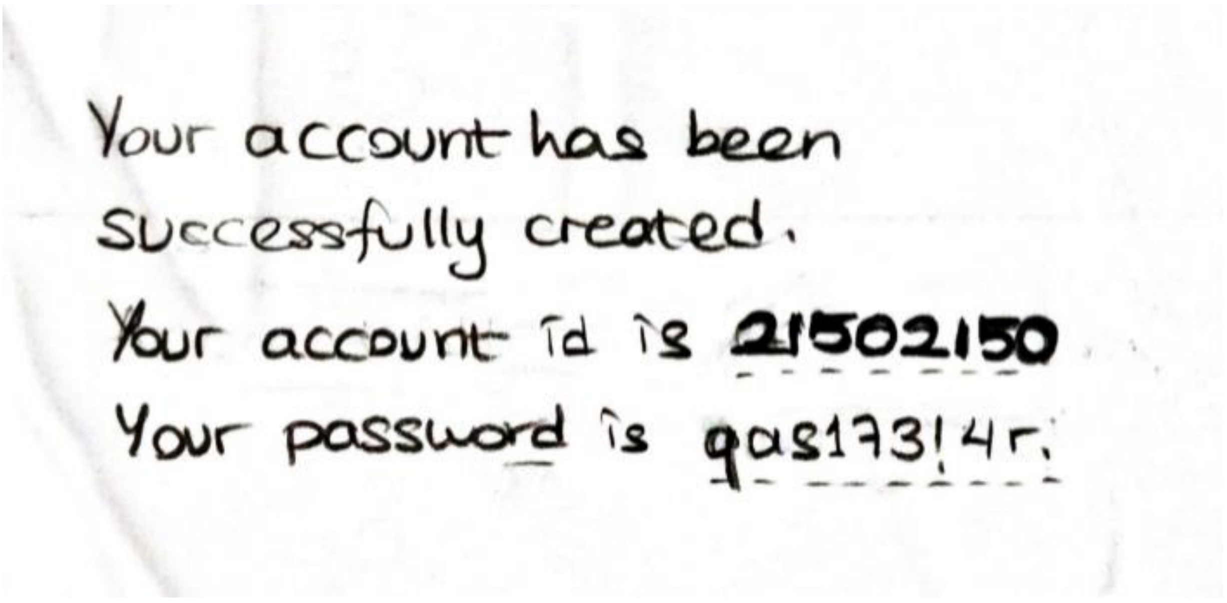
Register

● Player ○ Coach ○ President ○ Agent

Name:	Batu	Prefered Foot:	Right
Surname:	Orhon	Position:	Goalkeeper
Nationality:	Turkish	Birthday:	01.07.1996
Team Name:	Bilkent		

Create Account

Cancel



Your account has been successfully created.
Your account id is 21502150
Your password is qas173!4r.

Figure 1. Before Registration
Figure 2.After Registration
Inputs: @name, @surname, @pref_foot, @nationality, @position, @birthdate
Process: After user clicks "Register" button on the login page, the user is directed to this page. There are different types of users and the user chooses the appropriate user type. For each user type, the web page and the fields of the registration form will change, e.g. Player will have preferred foot whereas coach will have start date. For simplicity, only player's registration mockup is shown above. When user clicks the "Create Account" button after filling appropriate fields, an account_id and password are generated by the system, then an account is created in its corresponding table. After the entry of the account type is created, a new page is shown to the user with his/her account id and password. Below are sql queries which will execute if the user has chosen the respective user type:

Note: Account id and password, which are generated in the backend are referenced as @account_id and @password respectively.

@name: Name of the person linked to the account
@surname: Surname of the person linked to the account
@pref_foot: Preferred foot of the player, can be left or right
@nationality: National origin of the player
@position: Main position of the player
@birthdate: Birthdate of the player

5.3.1. Player Registration

SQL Statements:
Insert a player entry to the db:

INSERT INTO player VALUES(@account_id, @password, @name, @surname, @nationality, @birthdate, @kit_no, @pref_foot, @recovery_date, @suspend_date, @belong_to_team_name);

5.3.2. Coach Registration

SQL Statements:

Insert a coach entry to the db:

INSERT INTO coach VALUES(@account_id, @password, @name, @surname, @nationality, @birthdate, @start_date, @team_name);

5.3.3. President Registration

SQL Statements:

Insert a president entry to the db:

INSERT INTO president VALUES(@account_id, @password, @name, @surname, @nationality, @birthdate, @start_date, @team_name);

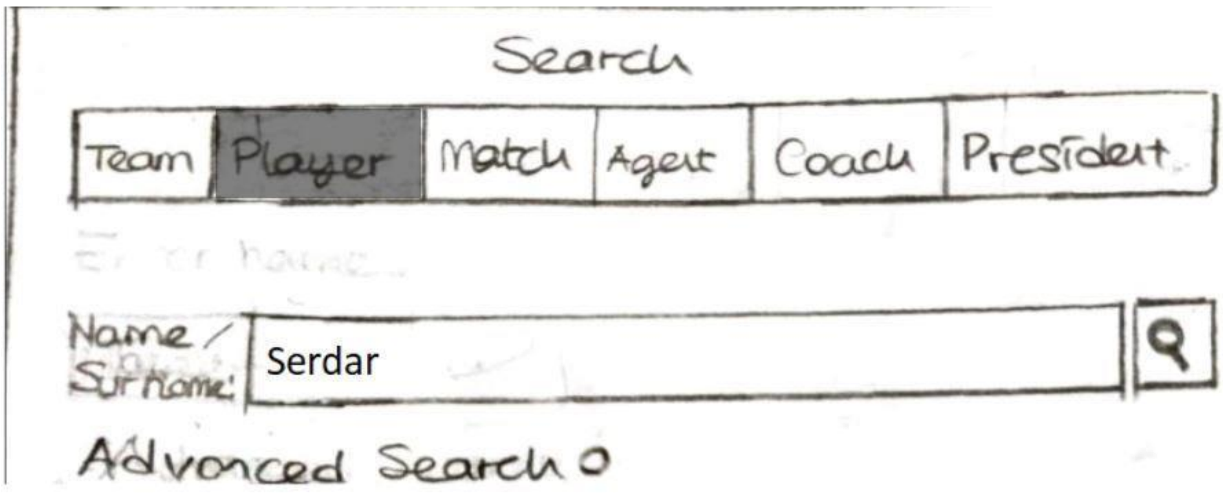
5.3.4. Agent Registration

Insert an agent entry to the db:

SQL Statements:

INSERT INTO agent VALUES(@account_id, @password, @name, @surname, @nationality, @birthdate, @player_account_id);

5.4. BASIC SEARCH PAGE



Inputs: @table, @name, @account_id

Process: User may select a search target among the predefined options and provide a name. Then, upon pressing Search button, a search query in the database is executed and the results are shown on the screen.

@table: Determines the table in which the query will be executed

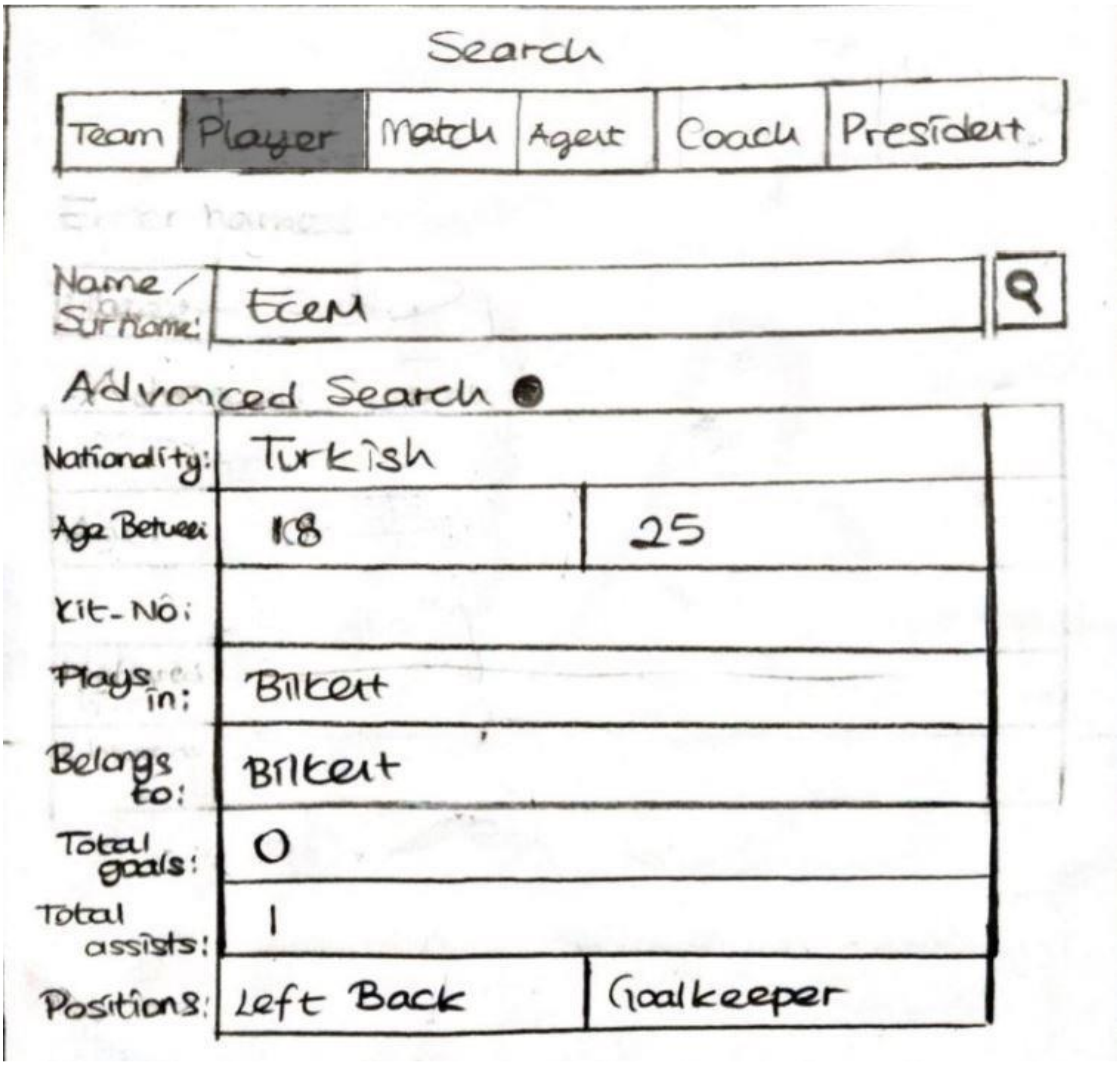
@name: Name of the player
@account_id: Extracted from @name, @account_id is created by the system and added into the query in case it is needed.

SQL Statements:
Retrieve all from the target table with name = @name:
SELECT *
FROM @table
WHERE name = @name;

5.5. ADVANCED SEARCH PAGE

In this part, mockups shown below correspond to extreme cases where the user selects most advanced search options. Generally, the actual search query will be generated on the back-end side by first validating the search option and concatenating it to the search query. Therefore, a search query with null values, such as "... where account_id = null …" is eliminated automatically.

5.5.1. SEARCH AMONG PLAYERS



Name / Surname:	Ecem	
Nationality:	Turkish	
Age Between:	18	25
Kit_No:		
Plays in:	Bilkent	
Belongs to:	Bilkent	
Total goals:	0	
Total assists:	1	
Positions:	Left Back	Goalkeeper

Inputs: @name, @nationality, @age_lower, @age_higher, @kit_no, @plays_in_team_name, @belongs_to_team_name, @total_goals_min, @total_assists_min, @position_main, @position_sec

Process: When user clicks "Search" button on the main page, user will be directed to search section. There, user will be asked to choose a category to search from. For each category, there will be different fields available. In players section, default values for string variables are empty strings; 0 and 90 for age limits respectively and 0 for goal and assist lower bounds.

@name: Name of the player
@nationality: National origin of the player
@age_lower: lowest age of the players selected from the database
@age_higher: highest age of the players selected from the database
@kit_no: Kit no of the player

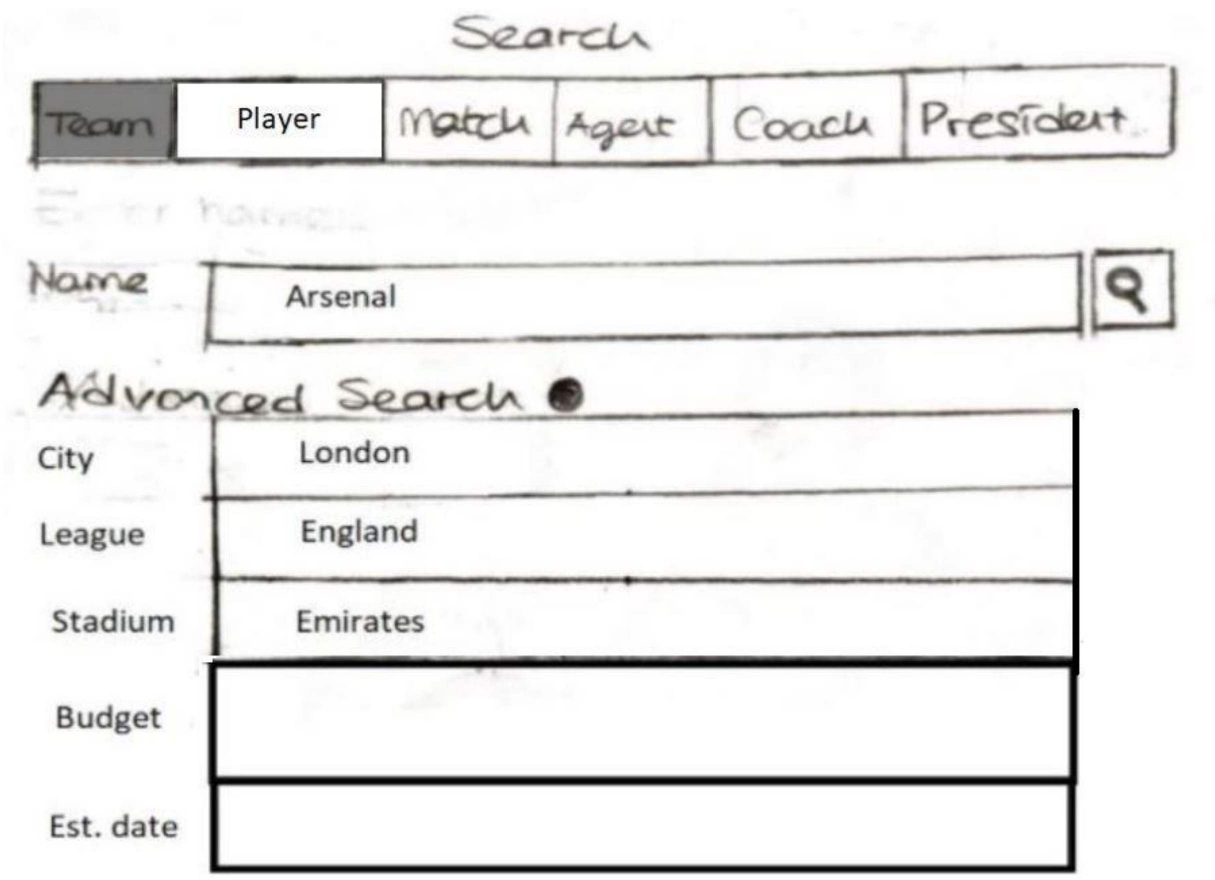
@plays_in_team_name: Name of the team the player currently plays in
@belongs_to_team_name: Name of the team the player belongs to
@total_goals_min: Minimum number of goals the player should have scored
@total_assists_min: Minimum number of assists the player should have made
@position_main: Main position of the player
@position_sec: Secondary position of the player

SQL Statements:
Example of an advanced search with @kit_no as null, therefore the query is generated without searching @kit_no, in order to provide players with any kit number who confirm to rest of the restrictions:

```
WITH stats(player_id, goal_count, assist_count) as (
        SELECT pg.account_id, pg.sum_goal, pa.assist_sum
        FROM player_goal as pg, player_assist as pa
        WHERE pg.account_id = @account_id and pa.account_id = @account_id)
SELECT*
FROM player natural join stats, position
WHERE (charindex(@name, name) >= 0 or charindex(@name, surname) >= 0) and
        age() >= @age_lower and age() <= @age_higher and
        charindex(@plays_in_team_name, plays_in_team_name) >= 0 and
        goal_count >= @total_goals_count and
        assist_count >= @total_assists_count and
        (position=@position_main or position=@position_sec or @position=null or
        @position_sec=null);
```

NOTE: In the mockup, input taken from the user is a number corresponding the age of the player. In back-end, we convert that age to birthdate and insert birthdate instead of age.

5.5.2. SEARCH AMONG TEAMS



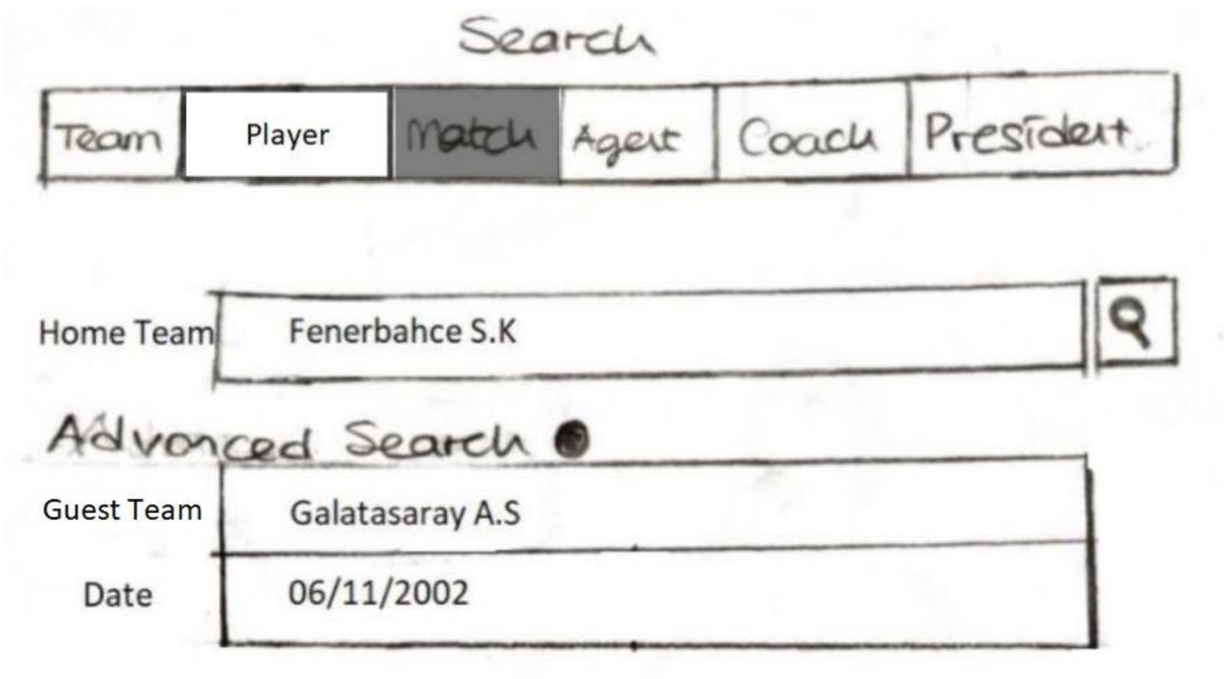
Inputs: @team_name, @city, @league_place, @stadium_name, @budget, @est_date
Process: In case user clicks "Search" button on the main page, user will be directed to search section. There, user will be asked to choose a category to search from. For each category, there will be different fields available. In team section, default values for string variables are empty strings.

@team_name: Name of the team
@city: City of the team
@league_place: Rank of the team in the league
@stadium_name: Name of the stadium of the team
@budget: Total budget of the team
@est_date: Establishment date of the team

SQL Statements:
An example of advanced search among teams with @budget and @est_date equal to null, therefore the query is generated without any constraint on them:

```
SELECT *
FROM teams NATURAL JOIN president
WHERE (charindex(@team_name, team_name) >= 0 AND
        charindex(@city, city) >= 0 AND
        charindex(@stadium_name, stadium_name) >= 0 AND
        charindex(@president, president.name) >= 0 AND
        charindex(@league_name, league_name) >= 0 AND
        charindex(@coach_name, coach_name) >= 0 AND
        (@budget <= budget) AND
        (datediff(@est_date, establishment_date)>= 0 OR
         @est_date=null)
);
```

5.5.3. SEARCH AMONG MATCHES

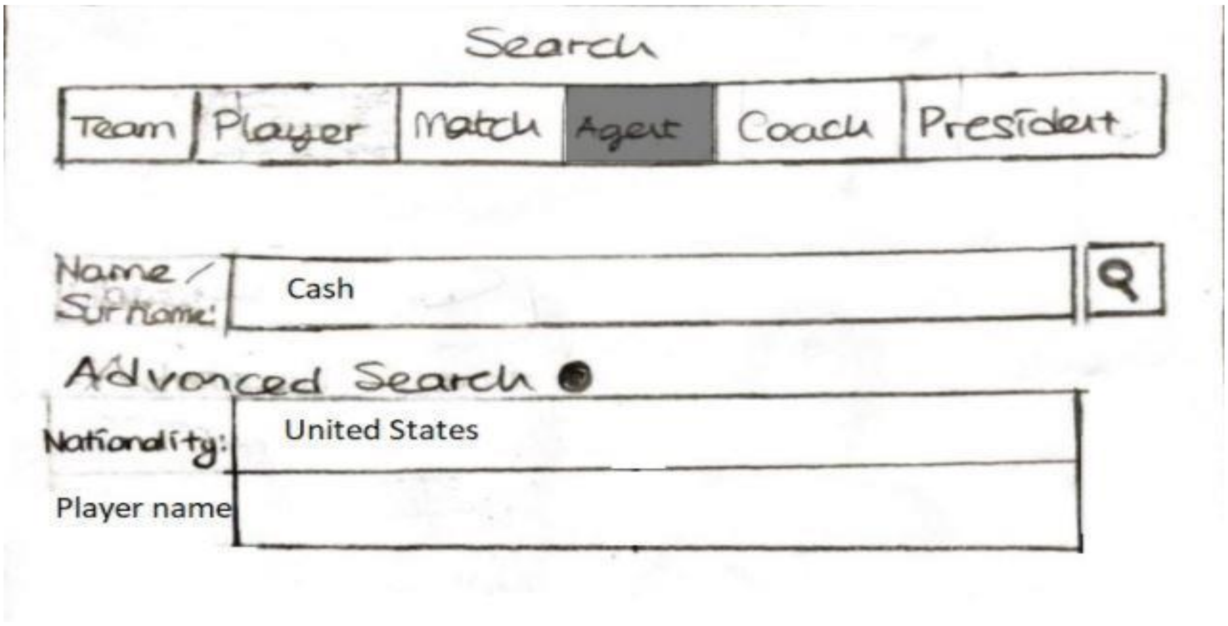


Inputs: @home_team_name, @guest_team_name, @date
Processes: If user clicks “Search” button on the main menu and chooses “Match” option, user will be prompted with that screen. User can search matches using home team name, away team name and date of the match. However, some of these fields might be left empty and these fields will not be added to searches.
@home_team_name: Home team of the matchup
@guest_team_name: Guest team of the matchup
@date: Date of the matchup

SQL Statements:
Advanced search among matchups where none of the inputs are null (they could have been null, but that would exclude them from the generated query):

```
SELECT *
FROM match
WHERE (charindex(home_team_name, @home_team_name) >= 0 AND
          (charindex(guest_team_name, @guest_team_name) >= 0 AND
          date=@date
);
```

5.5.4. SEARCH AMONG AGENTS



Inputs: @name, @nationality, @player_name

Processes: If user chooses to search among agents, they will proceed to this page. In this page, the user can make search using agent's name or surname, player's name or surname and nationality of the agent. Some of these fields might be left blank and in that case, these fields will not be taken into consideration while doing searches.

@name: Name of the agent

@nationality: National origin of the agent

@player_name: A substring of name or surname of the player agent represents

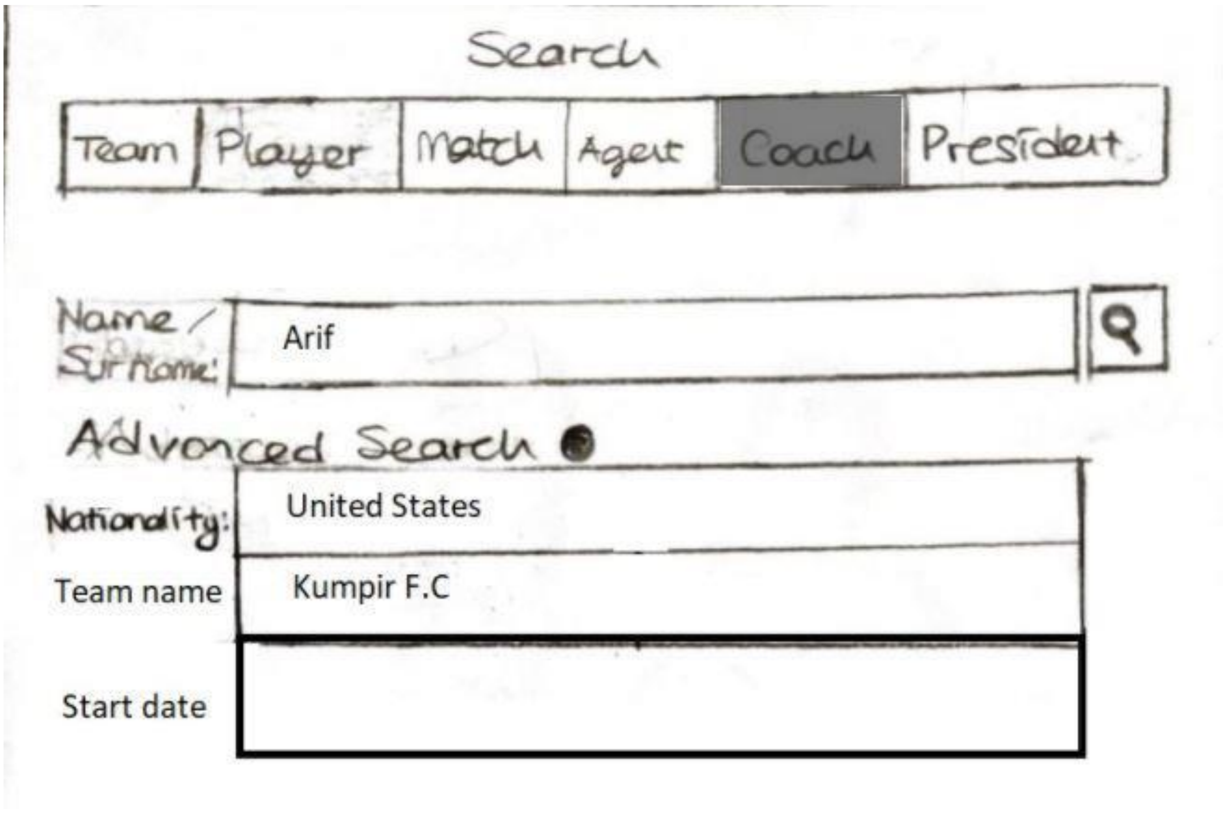
SQL Statements:

Find the agent with a given nationality, name, or player name-surname substring he represents:

```
select agent.name, agent.surname, agent.nationality, agent, player.name, player.surname

from agent, player

WHERE (player.account_id = agent.player_account_id

        AND (charindex(@name, agent.name) >= 0

        OR charindex(@name, agent.surname) >= 0)

        AND (charindex(@player_name, player.name) >= 0

        OR (charindex(@player_name,  player.surname) >= 0)

        AND (@nationality=agent.nationality)

);
```

NOTE: In the mockup, input taken from the user is a number corresponding the age of the player. In back-end, we convert the age to birthdate and insert birthdate instead of age.

5.5.5. SEARCH AMONG COACHES



Inputs: @coach_name, @nationality, @team_name, @start_date

Processes: If user chooses to search among coach, they will proceed to this page. In this page, the user can make search using coach's name or surname, team's name and starting date of the agent. Some of these fields might be left blank and in that case, these fields will not be taken into consideration while doing searches.

@coach_name: Name of the coach

@nationality: National origin of the coach

@team_name: Name of the team the coach is working for

@start_date: Start date of coach's contract

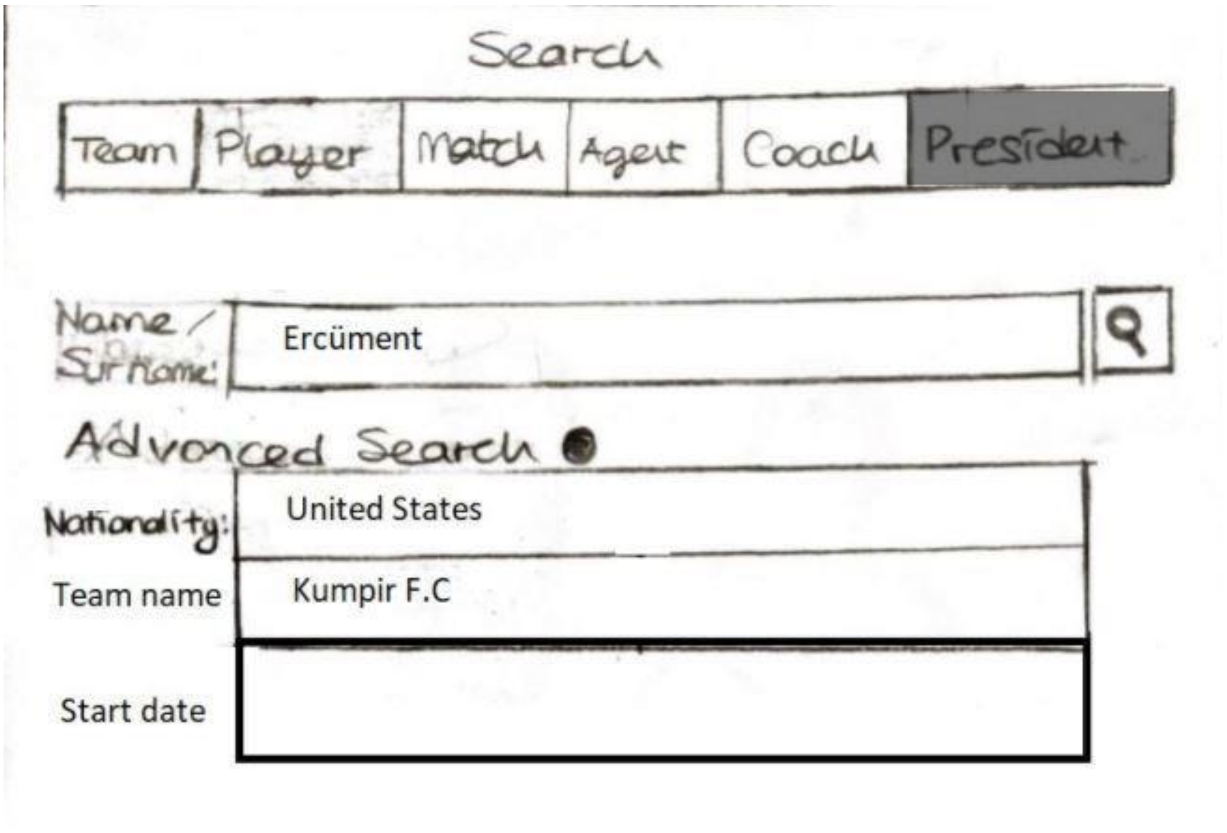
SQL Statements:

Find a coach with either with a given name, nationality, the team he works for, or contract start date:

```
search *

from coach

where ((charindex(@coach_name, coach.name) >= 0 or charindex(@coach_name,
coach.surname) >= 0) and

        (@nationality=coach.nationality) and

        (@start_date=coach.start_date)

);
```

Note: Since start date is assumed to be null in this case, it is excluded from the query.

5.5.6. SEARCH AMONG PRESIDENTS



Inputs: @president_name, @nationality, @team_name, @start_date

Processes: If user chooses to search among presidents, they will proceed to this page. In this page, the user can make search using president's name or surname, team's name and starting date of the president. Some of these fields might be left blank and in that case, these fields will not be taken into consideration while doing searches.

@president_name: Name of the president subject to search

@nationality: Nationality of the president subject to search

@team_name: Team name of the president subject to search

@start_date: Starting date of the president subject to search

SQL Statements:

An example of advanced search among presidents with @start_date equal to null, therefore the query is generated without any constraint on it:

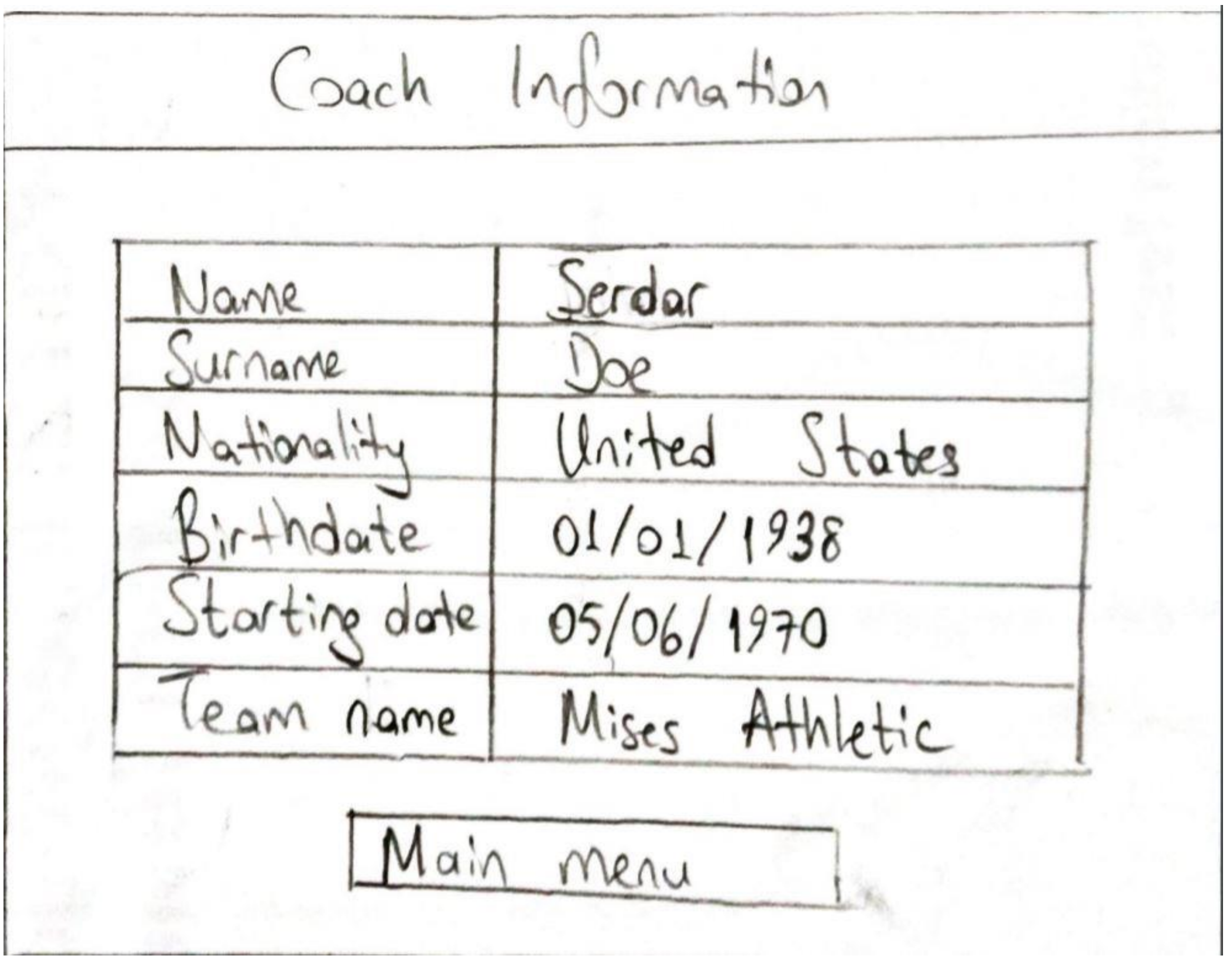
SELECT *

FROM president

WHERE ((charindex(@president_name, president.name) >= 0 or

```
            charindex(@president_name, president.surname) >= 0) and
            @nationality=president.nationality
);
```

5.6. COACH PAGE



Coach Information

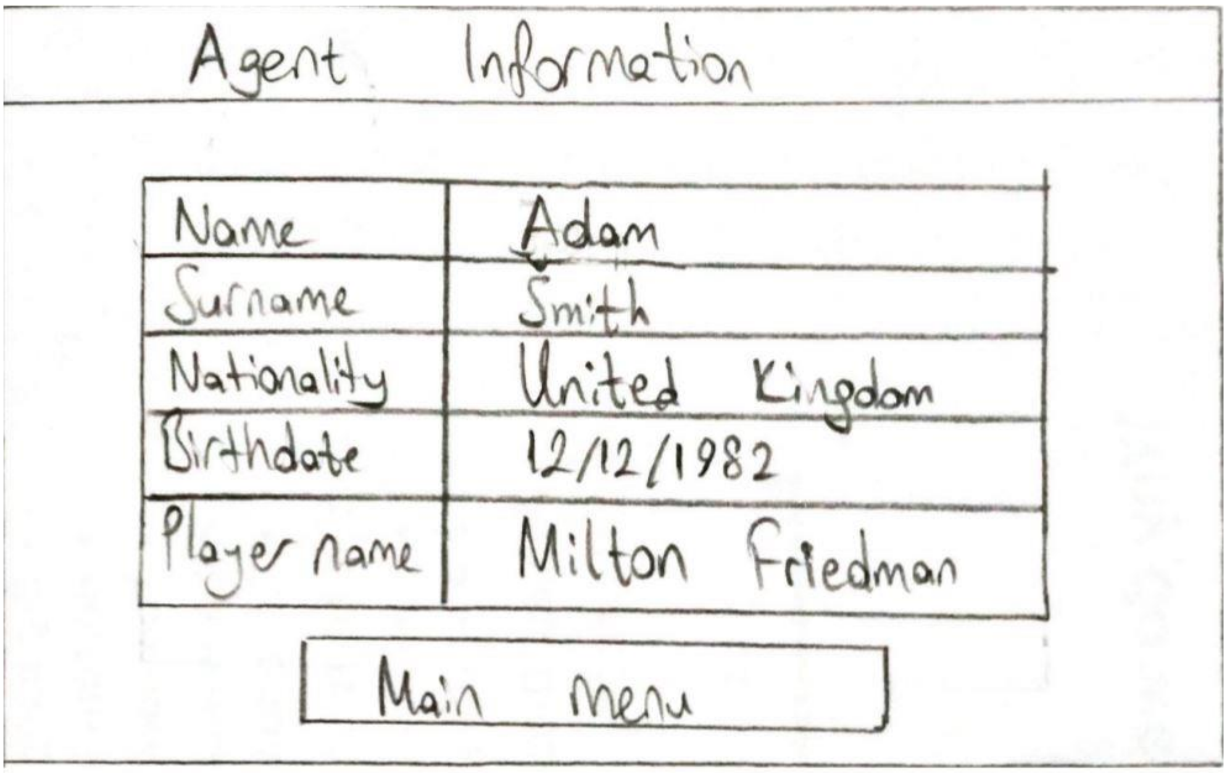
Name	Serdar
Surname	Doe
Nationality	United States
Birthdate	01/01/1938
Starting date	05/06/1970
Team name	Mises Athletic

Inputs: @account_id
Process: If a coach logs in to the system, their information will be displayed. From the login page, the account id will be received and the corresponding entry from the Coach table will be selected.
@account_id: Account id of the coach whose webpage is shown

SQL Statements:
Find a coach with given account_id:

```
select name, surname, nationality, birthdate, start_date, team_name
        from coach where account_id = @account_id;
```

5.7. AGENT PAGE

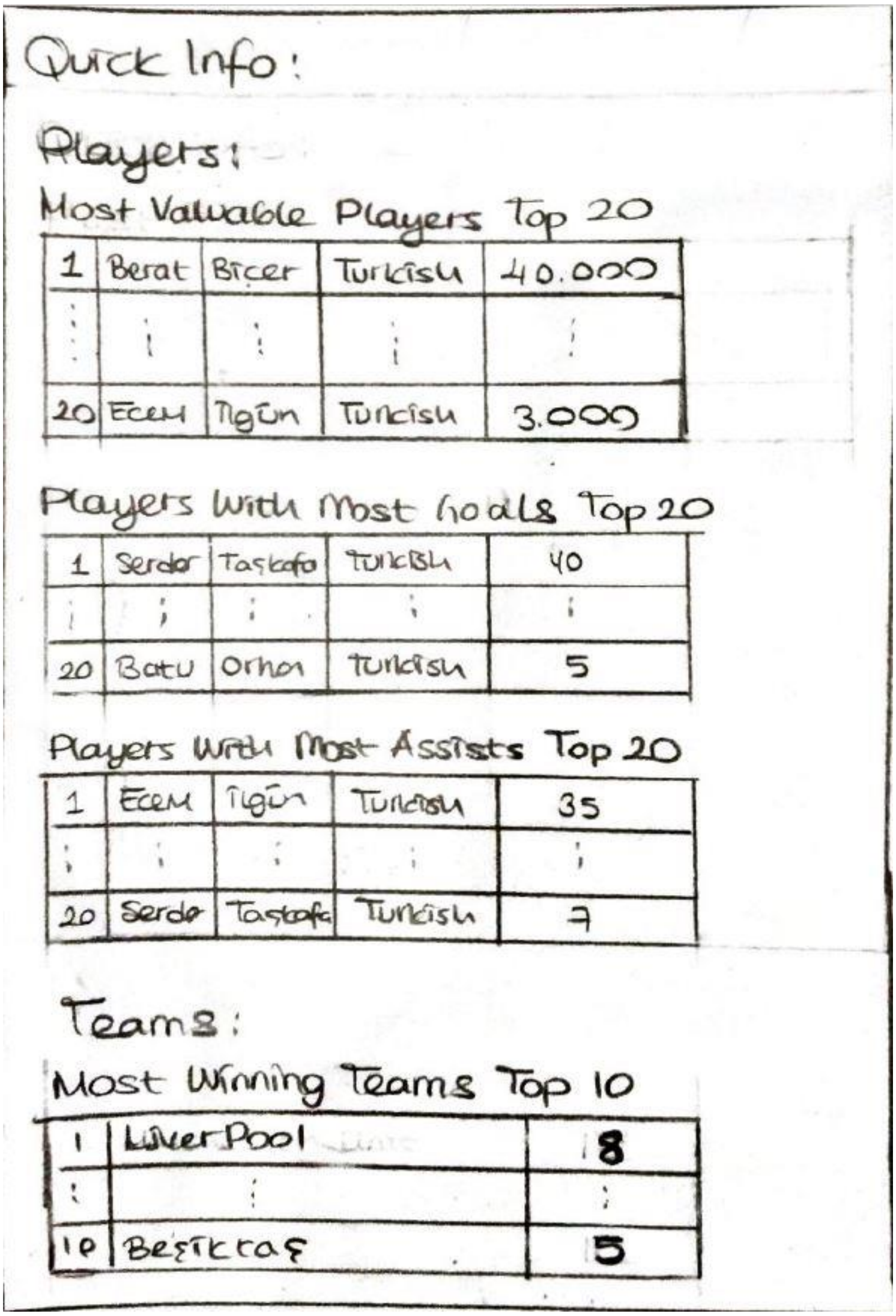


Name	Adam
Surname	Smith
Nationality	United Kingdom
Birthdate	12/12/1982
Player name	Milton Friedman

Inputs: @account_id
Process: If an agent logs in to the system, their information will be displayed. From the login page, the account id will be received and the corresponding entry from the Coach table will be selected. Since the pages for agents, coaches and presidents are similar, mockup only for the page for agents are added.
@account_id: Account id of the agent whose webpage is shown

SQL Statements:
Find an agent with given account_id:
select agent.name, agent.surname, agent.nationality, agent.birthdate, player.name, player.surname
    from agent, player
    where account_id = @account_id and player.account_id=player_account_id;

5.8. QUICK INFO



Quick Info:

Players:

Most Valuable Players Top 20

1	Berat	Bicer	Turkish	40.000
⋮	⋮	⋮	⋮	⋮
20	Ecem	Ilgün	Turkish	3.000

Players With Most Goals Top 20

1	Serdar	Taskafa	Turkish	40
⋮	⋮	⋮	⋮	⋮
20	Batu	Orhan	Turkish	5

Players With Most Assists Top 20

1	Ecem	Ilgün	Turkish	35
⋮	⋮	⋮	⋮	⋮
20	Serdar	Taskafa	Turkish	7

Teams:

Most Winning Teams Top 10

1	LiverPool	8
⋮	⋮	⋮
10	Beşiktaş	5

Inputs: None

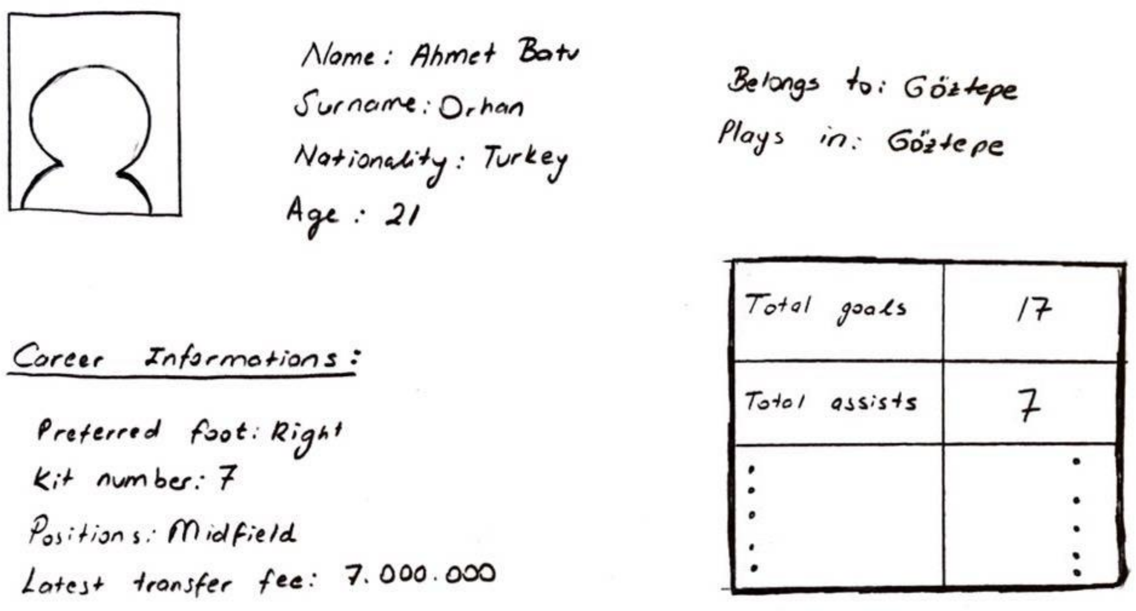
Process: User can see some selected statistics for public display from the database at this page.

SQL Statements:

Retrieving information from the views which we keep for quick info:

```
SELECT *
FROM most_valuable_20_players;
SELECT *
FROM top20_goals;
SELECT *
FROM top20_assists;
SELECT *
FROM most_winning_teams;
```

5.9. PLAYER



5.9.1. PUBLIC PLAYER PAGE

Inputs: @player_account_id

Process: When a public user or an unauthorized account enters a player's page, this page is displayed which contains generic, non-sensitive information.

@player_account_id: account id of the player of the webpage

SQL Statements:

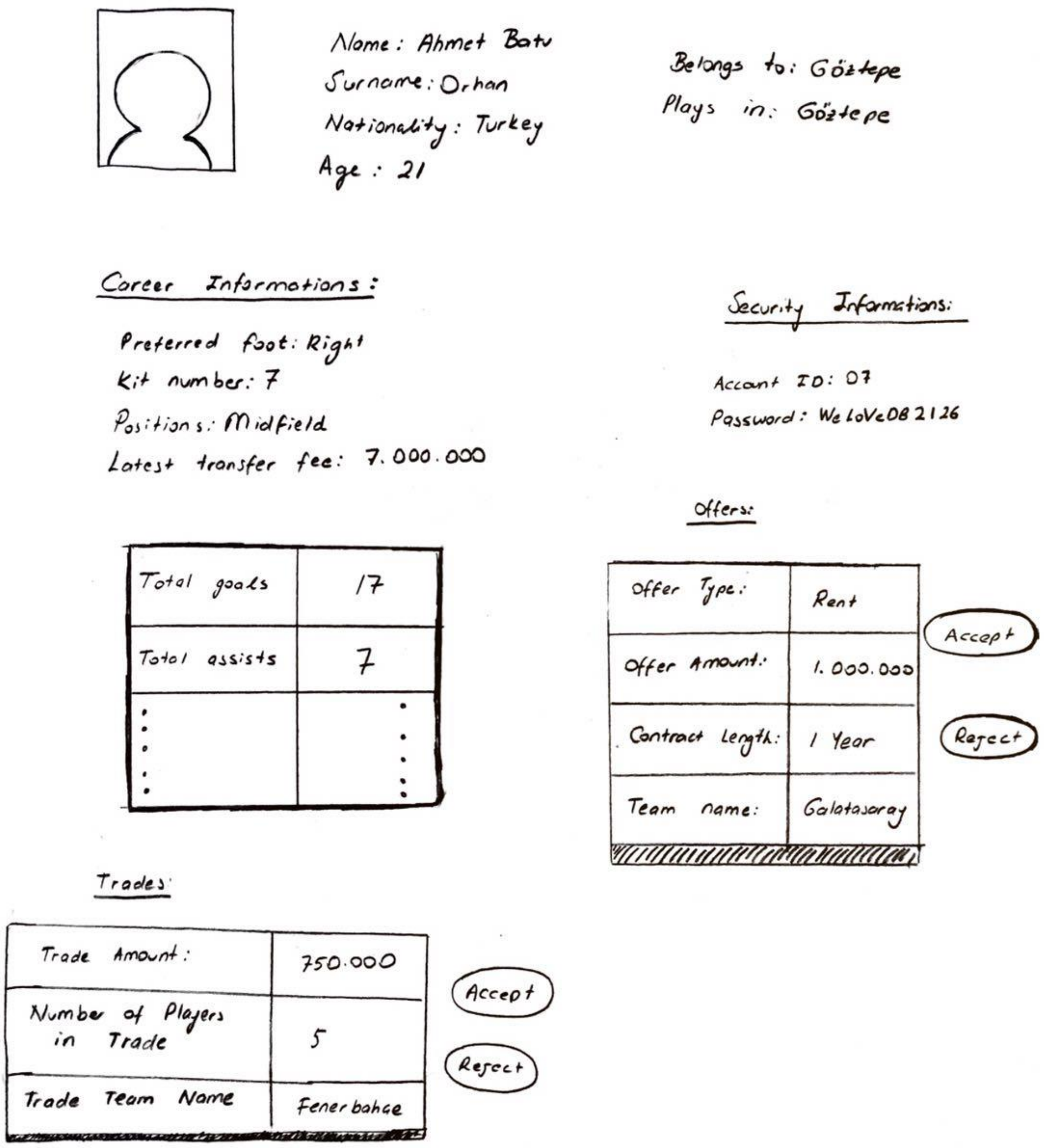
Retrieving basic player information:

```
SELECT pg.sum_goal, pa.sum_assist, p.account_id, p.belongs_to_team_id,
p.plays_in_team_id, p.name, p.surname, p.nationality, p.birthdate, p.pref_foot, p.kit_no,
p.prev_transfer_fee

FROM player AS p, player_goal AS pg, player_assist AS pa

WHERE pg.account_id = p.account_id AND pa.account_id = p.account_id AND p.account_id =
@player_account_id;
```

Retrieving player positions:

```
SELECT *

FROM position

WHERE account_id = @player_account_id;
```

5.9.2. PLAYER HOME PAGE



Name : Ahmet Batu
Surname : Orhan
Nationality : Turkey
Age : 21

Belongs to: Göztepe
Plays in: Göztepe

Career Informations:

Preferred foot: Right
Kit number: 7
Positions: Midfield
Latest transfer fee: 7.000.000

Total goals	17
Total assists	7
⋮	⋮

Security Informations:

Account ID: 07
Password: WeLoVeDB2126

Offers:

Offer Type:	Rent
Offer Amount:	1.000.000
Contract Length:	1 Year
Team name:	Galatasaray

Trades:

Trade Amount:	750.000
Number of Players in Trade	5
Trade Team Name	Fenerbahce

Inputs: @player_account_id

Process: When an authorized player account enters the its own page, player will able the see all the sensitive and non sensitive information.

@player_account_id: Account id of the player who has signed in

SQL Statements:

Retrieving basic player information:

```
SELECT pg.sum_goal, pa.sum_assist, p.account_id, p.belongs_to_team_id,
p.plays_in_team_id, p.name, p.surname, p.nationality, p.birthdate, p.pref_foot, p.kit_no,
p.prev_transfer_fee, p.account_id, p.password.

FROM player AS p, player_goal AS pg, player_assist AS pa

WHERE pg.account_id = p.account_id AND pa.account_id = p.account_id AND p.account_id =
@player_account_id;
```

Retrieving player positions:

```
SELECT *

FROM position

WHERE account_id = @player_account_id;
```

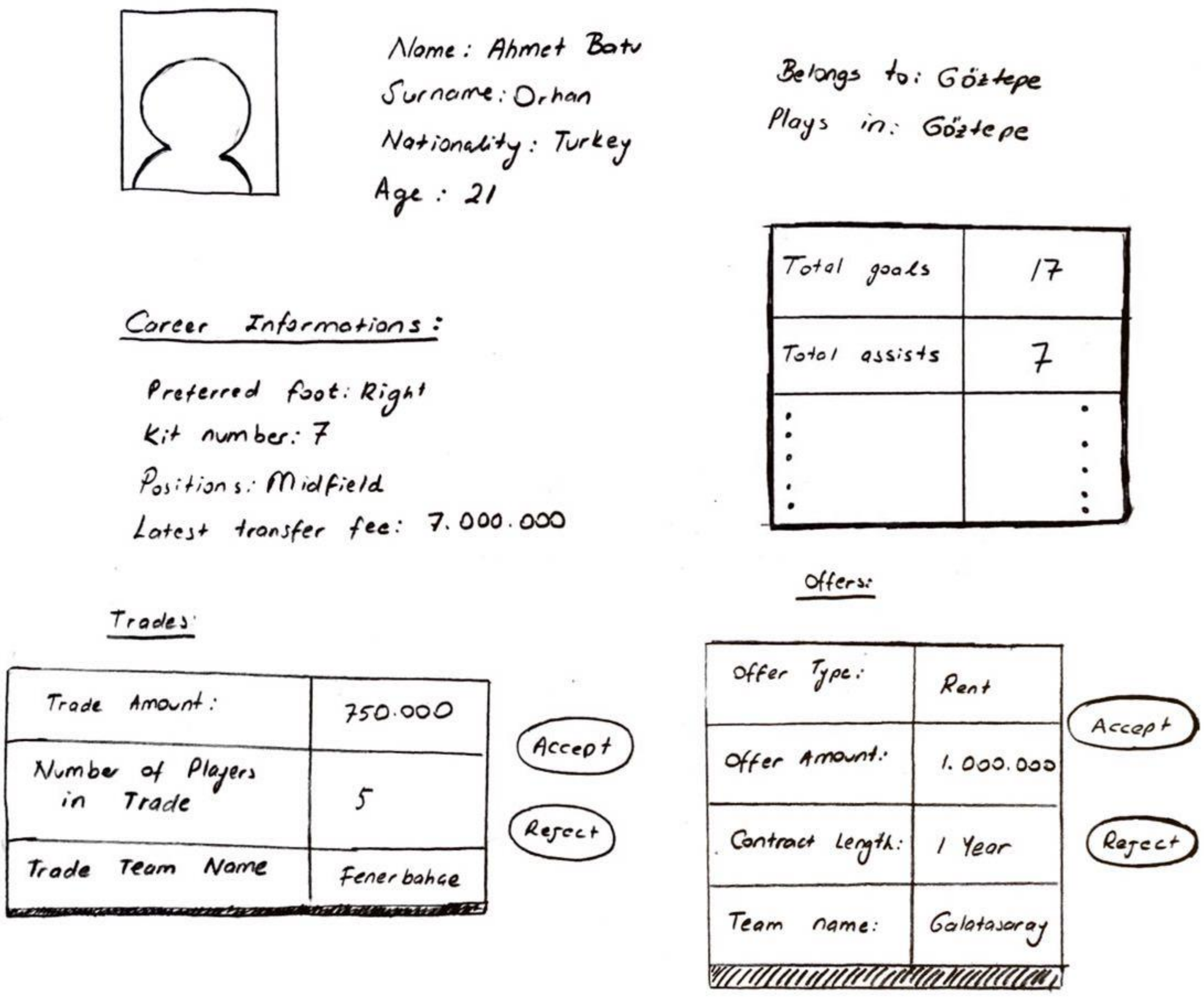
Retrieving player’s ongoing transfer/rent requests:

```
SELECT op.offer_type, op.offer_amount, op.offer_contract_end, op.offer_team_name

FROM player AS p, propose_offer AS po, offers_player AS op

WHERE p.account_id = @player_account_id AND

        p.account_id = po.player_account_id AND

        po.offer_id = op.offer_id;
```

Retrieving player’s ongoing trade requests:

```
SELECT tp.trade_amount, tp.trade_no_players, tp.trade_team_name

FROM player AS p, propose_trade AS pt, trades_player AS tp

WHERE p.account_id = @player_account_id AND

        p.account_id = pt.player_account_id AND

        pt.trade_id = tp.trade_id;
```

5.9.3. PLAYER PAGE - AUTHENTICATED TO PRESIDENT



Name : Ahmet Batu
Surname : Orhan
Nationality : Turkey
Age : 21

Belongs to: Göztepe
Plays in: Göztepe

Total goals	17
Total assists	7
⋮	⋮

Career Informations :

Preferred foot: Right
Kit number: 7
Positions: Midfield
Latest transfer fee: 7.000.000

Offers:

Offer Type:	Rent
Offer Amount:	1.000.000
Contract length:	1 Year
Team name:	Galatasaray

Accept
Reject

Trades:

Trade Amount:	750.000
Number of Players in Trade	5
Trade Team Name	Fenerbahce

Accept
Reject

Inputs: @player_account_id

Process: When an authorized president account enters the its one of the players' page which belongs to the presidents team, president will able the see all the sensitive and non-sensitive informations except the players account ID and password.

@player_account_id: account id of the player of the webpage

SQL Statements:

Retrieving basic player information:

```
SELECT pg.sum_goal, pa.sum_assist, p.account_id, p.belongs_to_team_id,
p.plays_in_team_id, p.name, p.surname, p.nationality, p.birthdate, p.pref_foot, p.kit_no,
p.prev_transfer_fee

FROM player AS p, player_goal AS pg, player_assist AS pa

WHERE pg.account_id = p.account_id AND

        pa.account_id = p.account_id AND

        p.account_id = @player_account_id;
```

Retrieving player positions:

```
SELECT *

FROM position

WHERE account_id = @player_account_id;
```

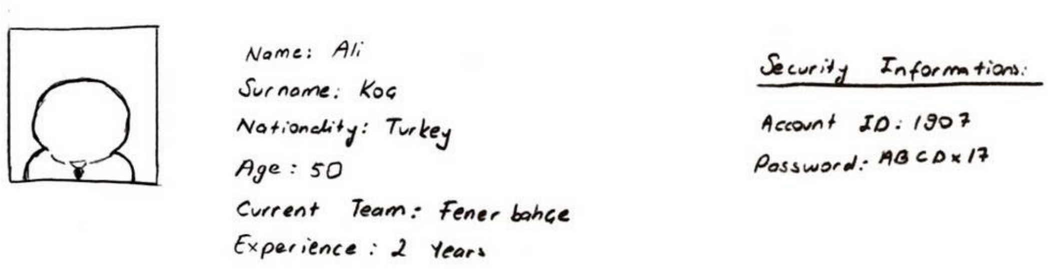
Retrieving player's ongoing transfer/rent requests:

```
SELECT op.offer_type, op.offer_amount, op.offer_contract_end, op.offer_team_name

FROM player AS p, propose_offer AS po, offers_player AS op

WHERE p.account_id = @player_account_id AND

        P.account_id = po.player_account_id AND

        po.offer_id = op.offer_id;
```

Retrieving player's ongoing trade requests:

```
SELECT tp.trade_amount, tp.trade_no_players, tp.trade_team_name

FROM player AS p, propose_trade AS pt, trades_player AS tp

WHERE p.account_id = @player_account_id AND

        p.account_id = pt.player_account_id AND

        pt.trade_id = tp.trade_id;
```

5.9.4. PUBLIC PRESIDENT PAGE



Name: Ali

Surname: Koç

Nationality: Turkey

Age: 50

Current Team: Fener bahçe

Experience: 2 Years

Security Informations:

Account ID: 1907

Password: ABCDx17

Inputs: @president_account_id

Process: When an unauthorized user enters the a president's page, this user will able the see only the nonsensitive informations.

@president_account_id: account id of the president of the webpage

SQL Statements:

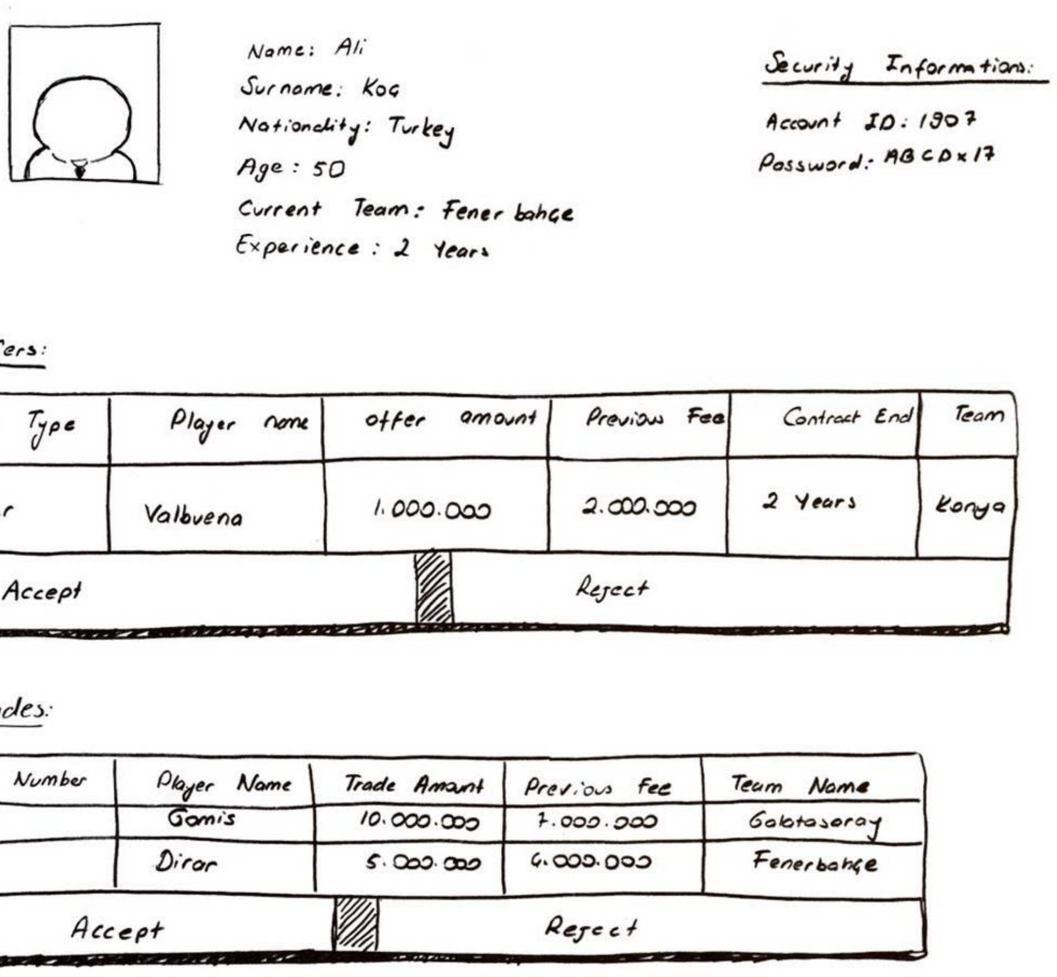
Retrieving basic president information:

SELECT name, surname, nationality, birthdate, start_date, team_name

FROM president

WHERE account_id = @president_account_id

5.9.5. PRESIDENT HOME PAGE



Name: Ali
Surname: Koç
Nationality: Turkey
Age : 50
Current Team: Fener bahçe
Experience : 2 Years

Security Informations:
Account ID: 1907
Password: ABCDx17

Offers:

Offer Type	Player name	offer amount	Previous Fee	Contract End	Team
Transfer	Valbuena	1.000.000	2.000.000	2 Years	Konya

Trades:

Trade Number	Player Name	Trade Amount	Previous fee	Team Name
1	Gomis	10.000.000	7.000.000	Galatasaray
1	Diror	5.000.000	6.000.000	Fenerbahçe

Inputs: @president_account_id

Process: When an authorized president enters its own page, this user will able the see not only the nonsensitive informations but also the sensitive informations.

@president_account_id: Account id of the president who has signed in

SQL Statements:

Retrieving basic president information:

```
SELECT account_id, password, name, surname, nationality, birthdate, start_date, team_name

FROM president

WHERE account_id = @president_account_id;
```

Retrieving ongoing transfer/rent requests involving players who belong to the team of the president:

```
SELECT ot.offer_type, ot.player_name, ot.offer_amount, ot.prev_transfer_fee,
ot.pffer_contract_end, ot.offer_team_name

FROM president AS p, propose_offer AS po, offers_team AS ot

WHERE p.account_id = po.president_account_id AND

        po.offer_id = ot.offer_id;
```

Retrieving ongoing trade requests involving players who belong to the team of the president:

```
SELECT tt.trade_id, tt.player_name, tt.trade_amount, tt.prev_transfer_fee,
tt.trade_team_name
FROM president AS p, propose_trade AS pt, trades_team AS tt
WHERE p.account_id = pt.president_account_id AND

        pt.trade_id = tt.trade_id;
```

6. ADVANCED DATABASE COMPONENTS

6.1. VIEWS

6.1.1. MOST VALUABLE PLAYERS TOP 20 VIEW

This view is used to display the top 20 most valuable players of the league in Quick Info page.

```
CREATE VIEW most_valuable_20_players (rank, name, surname, nationality,
prev_transfer_fee)
AS SELECT *
FROM (SELECT RANK() OVER(ORDER BY prev_transfer_fee DESC) AS fee_rank,
                name, surname, nationality, prev_transfer_fee
          from player
          ORDER BY fee_rank)
WHERE fee_rank < 21;
```

6.1.2. PLAYERS WITH MOST GOALS TOP 20 VIEW

This view is used to display top 20 most goal scoring players of the league. Also, a version of the view top20_goals, player_goal, is used in advanced search, if the user specifies a constraint on total number of goals scored by the player.

```
CREATE VIEW top20_goals (rank, name, surname, nationality, total_goals)
AS SELECT  grouped_goals.rank, player.name, player.surname, player.nationality,
grouped_goals.goal_sum
FROM player JOIN (
          SELECT RANK() OVER(ORDER BY sum(value) DESC) AS rank,
                    sum(value) AS goal_sum, account_id
          FROM statistics
          GROUP BY account_id
          HAVING type = "goal") AS grouped_goals
WHERE player.account_id = grouped_goals.account_id and grouped_goals.rank < 21
```

```
CREATE VIEW player_goal(account_id, sum_goal) AS

SELECT account_id, sum(value) AS goal_sum

FROM statistics GROUP BY account_id HAVING type = "goal";
```

6.1.3. PLAYERS WITH MOST ASSISTS TOP 20 VIEW

This view is used to display top 20 most assist making players of the league. Also, a version of the view top20_assists, player_assist, is used in advanced search, if the user specifies a constraint on total number of assists made by the player.

```
CREATE VIEW top20_assists (rank, name, surname, nationality, total_assists)
AS SELECT RANK() OVER(ORDER BY t as DESC) AS rank, p.name, p.surname, p.nationality,
t.assist_sum
FROM player AS p, (SELECT account_id, sum(value) as assist_sum
        FROM statistics
        WHERE type = “assist”
        GROUP BY account_id) as t
WHERE rank < 21 AND p.account_id = t.account_id;

CREATE VIEW player_assist (account_id, assist_sum) AS
        SELECT account_id, sum(value) as assist_sum
        FROM statistics
        WHERE type = “assist”
        GROUP BY account_id;
```

6.1.4. TEAMS THAT WON THE MOST MATCHES TOP 10 VIEW

This view is used to display top 10 most winning teams of the league, on Quick Info page.

```
CREATE VIEW most_winning_teams(win_rank, winner_team, win_count)
AS SELECT RANK() OVER(ORDER BY win_count DESC) AS win_rank, winner_team, win_count
FROM (SELECT mtg1.team_name as winner_team, count * AS win_count
        FROM match_team_goalsum as mtg1, match_team_goalsum AS mtg2
        GROUP BY  mtg1.team_name
        HAVING mtg1.match_id = mtg2.match_id and mtg1.goal_sum > mtg2.goal_sum)
WHERE win_rank < 11;
```

6.1.5. GOALS PER MATCH FOR EACH TEAM VIEW

This view will be used in choosing the teams that won the most matches.

```
CREATE VIEW match_team_goalsum(match_id, team_name, goal_sum)
AS SELECT  match_id, team_name, sum(value) AS goal_sum
FROM statistics
GROUP BY match_id, team_id
HAVING type = “goal”;
```

6.1.6. PLAYER’S ONGOING TRANSFERS & TRADES & RENT DECISIONS VIEW

This view displays the offer and trade requests made on a specific player, to be displayed to that user. He then may choose to approve or disapprove the request.

```
CREATE VIEW offers_player (offer_id, offer_type, offer_amount, offer_contract_end,
offer_team_name)
AS SELECT o.offer_id, o.offer-type, o.offer-amount, o.contract-end, tm.team_name
FROM offer AS o, team AS tm, (SELECT offer_id, president_account_id
        FROM propose_offer
        WHERE player_account_id = @account_id) AS tab
WHERE tm.team_name = (
        SELECT p.team_name
        FROM president AS p
        WHERE p.account_id = tab.president_account_id) AND o.offer_id = tab.offer_id;

CREATE VIEW trades_player(trade_id, trade_amount, trade_no_players, trade_team_name)
AS SELECT t.trade_id, t.trade_amount, t.num_players, tm_team_name
FROM trade as t, team as tm, (SELECT trade_id, president_account_id
                              FROM propose_offer
                              WHERE player_account_id = @account_id) as tab
WHERE tm.team_name = (
        SELECT p.team_name
        FROM president as p
        WHERE p.account_id = tab.president_account_id) and t.offer_id = tab.offer_id;
```

6.1.7. PRESIDENT'S ONGOING TRANSFERS & TRADES & RENT DECISIONS VIEW

This view displays the offer and trade requests made on all the players of a team, and the president of that team is able to see them. He then may choose to approve or disapprove the request.

```
CREATE VIEW offers_team (offer_id, offer_type, player_name, offer_amount,
prev_transfer_fee, offer_contract_end, offer_team_name)
AS SELECT offer.offer_id, offer.offer-type, player.player_name, offer.offer-amount,
player.prev_transfer_fee, offer.contract-end, player.team_name
FROM offer, player, propose_offer
WHERE decidepresident_account_id = @account_id AND
         offer.offer_id = propose_offer.offer_id AND
        propose_offer.player_account_id = player.account_id;

CREATE VIEW trades_team (trade_id, player_name,  trade_amount, prev_transfer_fee,
trade_team_name)
AS SELECT trade.trade_id, player.player_name, trade.trade_amount,
        player.prev_transfer_fee, player.team_name
FROM trade, propose_trade, player
```

```
WHERE decidepresident_account_id = @account_id AND
        trade.trade_id = propose_trade.trade_id AND
        propose_trade.player_account_id = player.account_id
```

6.2. STORED PROCEDURES

We require a stored procedure per each of the following cases:

- Number of players participating in a match should be known at the time the match is created. This is also connected to number of statistics tuples created for each player in a given match.
- When an offer/trade is created, it stays in the database for as long as the time it takes for participants to approve or reject that request. Any tuple in propose_offer or propose_trade implies existence of an offer, which should be removed once the request reaches conclusion.

Therefore, we shall use the following stored procedures:

- An engine which will determine number of players and identities of those who participates in a match, which also creates the statistics entries for the attending players and fill them accordingly
- Autoclean engine that will create and remove requests for propose_offer and propose_trade tables

6.3. REPORTS

6.3.1. MONTHLY INJURY REPORT

Presidents and coaches are able to get a monthly injury report, consists of injured players in their team.

```
CREATE VIEW injury_report AS

SELECT plays_in_team_id, belongs_to_team_id, name, surname, kit_no, recovery_date

FROM player

where recovery_date != “”;
```

6.3.2. MONTHLY GOALS SCORED

Presidents and coaches are able to get a monthly report of goals scored within the last month.

```
CREATE VIEW monthly_goals as

SELECT * from statistics

WHERE type = “goal” and TIMESTAMPDIFF(MONTH, NOW(), date) < 2;
```

6.3.3. MONTHLY ASSISTS MADE

Presidents and coaches are able to get a monthly report of assists made within the last month.

```
CREATE VIEW monthly_assists as

SELECT * from statistics

WHERE type = “assist” and TIMESTAMPDIFF(MONTH, NOW(), date) < 2;
```

6.4. TRIGGERS

- A trigger to update each player’s total number of goals and assists as new statistics tuples are inserted
- A trigger to remove tuples from propose_offer, decide_offer, propose_trade, decide_trade
- A trigger to update most valuable players in the system when a new transfer occur
- A trigger to update top 20 most goal scoring - assist making players list
- A trigger to update top 20 most match winning teams after each match
- A trigger to update goals per match for every team after each match
- A trigger to refresh offers for players and presidents as new requests are made

6.5. CONSTRAINTS

- Budget of the offer-making team should be more than the transfer cost.
- Player that offer is made to should not have been transferred in the past 6 months.
- A president cannot make an offer to players who play in his own team.
- If a player is rented, a transfer offer cannot be made to that player.
- An offer must include at least 2 players, 1 from offer-making president’s team and 1 from offer-receiving president’s team.
- If a player is rented, he may not be included in trade offers.
- There must be exactly 2 teams included in the offer. In other words, there must be exactly 2 distinct presidents involved.
- A player who is recently transferred, traded or rented is suspended and therefore cannot be included in trades.
- There must be at least 2 matches among each pair of teams in the database.
- Time between consecutive matches between the same pair must be greater than or equal to 1 month
- If the capacity of the home team’s stadium is less than 10.000 people, a new stadium must be selected for the match.
- Each statistics tuple must be created by the admin within 3 days of the match date
- Number of distinct type of statistics for each player in each match is fixed. For example, for each player in a match there must be exactly N distinct statistics type as goal, assist, etc. where N is a predetermined number.

7. IMPLEMENTATION PLANS

- For front end HTML, CSS, Bootstrap, Javascript and JQuery,
- For back-end PHP,
- For database MySQL

will be used for implementation.

8. WEBSITE

Project Information: https://github.com/theyusko/database-group11

Project Website: cs353group11.bilkent.edu.tr