CS353 DATABASE SYSTEMS

PROJECT DESIGN REPORT

SOCIAL BETTING PLATFORM

Group 5

GROUP MEMBERS

ASUMAN AYDIN - 21502604

ECE ÇANGA - 21600851

ORKAN ÖZTRAK - 21703882

MELİSA ONARAN - 21301232

Project URL: https://mellonaran.github.io

Table of Contents

1 Revised E/R model

According to the feedback we had from the Proposal Report, we revised the E/R diagram of the project. We removed the redundant relationships and aggregation over some entities. We have added some entities and according to additions, we changed the relation between some entities.

1.1 Addition of Entities and Relations

For User, we have added Post Entity to keep his/her and other users' shares. Added relations between User and Post are like, share and create. Again for Users, Comment entities have been added, this is for the comment on the posts of users. Relation added for Comment and User is written. These are needed because data is higher than needed to keep it in one entity. Hence, we have decided to separate it.
To be able to keep the bet slips suggested by the editor as private to users, we have decided to create an entity called EditorBetSlips.

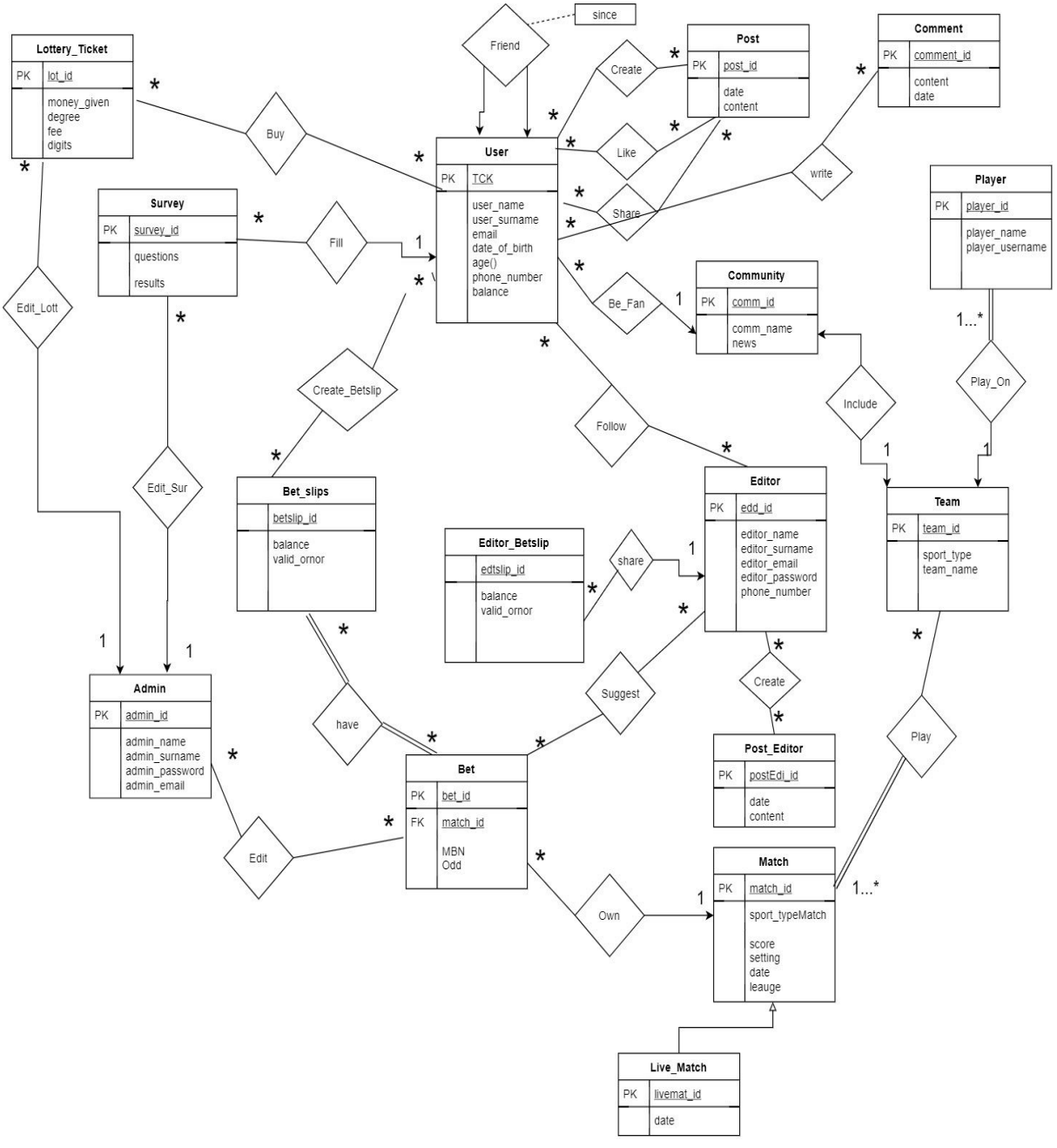
1.2 Removals

We have removed the aggregation over Bet, Match, and Team. After receiving feedback, we have decided to use more relations to keep dependencies between them and others. Also, we have removed the comment on the write relation of the editor because it is redundant to separate it from the Editor entity. Another removal is to choose the relation on the aggregation. With this removal, we have removed the sport type attribute on it. It is because these were redundant use of relations and attributes.

1.3 Changes

After adding Post and Comment entities, we changed the redundant relations between Betslip and User entities and replaced it with one relation which is create_Betslip. Also, Admin was connected to aggregation with update relation.

As the use of aggregation was wrong, we have changed the relation between Admin and Bets to Edit. In addition to Bets, as we have additional functionalities such as Lottery and Survey, Admin also edits these.



2 Relation Schemas

2.1 User

Relational Model:

user(TCK, username, user_name, user_surname, email, date_of_birth, age(), phone_number, balance);

Functional Dependencies:

TCK -> user_name, user_surname, date_of_birth, age(), phone_number, balance

email-> user_name, user_surname, date_of_birth, age(), phone_number, balance

username> user_name, user_surname, date_of_birth, age(), phone_number, balance

Candidate Keys:

{(TCK), (email),(username)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user(
        TCK varchar(11) PRIMARY KEY,
        username varchar(25) NOT NULL,
        user_name varchar(25),
        user_surname varchar(25),
        email varchar(30) NOT NULL,
        date_of_birth date,
```

age varchar (3),

phone_number int (11),

balance numeric (14,2));

2.2 Bet

Relational Model:

bet(bet_id, match_id, mbn, odd);

Functional Dependencies:

bet_id -> match_id, mbn, odd

Candidate Keys:

{(bet_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE bet(
        bet_id varchar (11) PRIMARY KEY,
        match_id varchar (11) NOT NULL,
        mbn int (9) NOT NULL,
        odd numeric(2,1) NOT NULL,
        FOREIGN KEY (match_id) REFERENCES match(match_id));
```

2.3 BetSlip

Relational Model:

bet_slip(betslip_id, balance, valid_or_not);

Functional Dependencies:

betslip_id -> balance, valid_or_not

Candidate Keys:

{(betslip_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE bet_slip(
        betslip_id varchar (11) PRIMARY KEY,
        balance int (12),
        valid_or_not bit(1) NOT NULL);
```

2.4 Match

Relational Model:

match(match_id, sport_type, score, setting, date, league);

Functional Dependencies:

match_id -> score, setting, date, league

Candidate Keys:

{(match_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE match(
        match_id varchar (11) PRIMARY KEY,
        sport_type int (2) NOT NULL,
        score numeric (4,2),
        setting varchar (21) NOT NULL ,
        date date NOT NULL,
        league int (2) NOT NULL);
```

2.5 Admin

Relational Model:

admin(admin_id, admin_name, admin_surname, admin_password, admin_email);

Functional Dependencies:

admin_id -> admin_name, admin_surname, admin_password

admin_email -> admin_name, admin_surname, admin_password

Candidate Keys:

{(admin_id),(admin_email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE admin(
        admin_id varchar (11) PRIMARY KEY,
        admin_name varchar (25) NOT NULL,
        admin_surname varchar (25),
        admin_password varchar (15) NOT NULL,
        admin_email varchar (30) NOT NULL
        );
```

2.6 Editor

Relational Model:

editor(ed_id, editor_name, editor_surname, editor_email, editor_password, phone_number);

Functional Dependencies:

ed_id -> editor_name, editor_surname, editor_password, phone_number

editor_email -> editor_name, editor_surname, editor_password, phone_number

Candidate Keys:

{(ed_id),(editor_email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor(
        ed_id varchar (11) PRIMARY KEY,
        editor_name varchar (25) NOT NULL,
        editor_surname varchar (25),
        editor_email varchar (30) NOT NULL,
        editor_password varchar (15) NOT NULL,
        phone_number int (11));
```

2.7 Team

Relational Model:

team(team_id, sport_type, team_name);

Functional Dependencies:

team_id -> sport_type, team_name

Candidate Keys:

{(team_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE team(
        team_id varchar (11) PRIMARY KEY,
        sport_type int (2) NOT NULL,
        team_name (30) NOT NULL);
```

2.8 Player

Relational Model:

player(player_id, player_name);

Functional Dependencies:

player_id -> player_name

Candidate Keys:

{(player_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE player(
        player_id varchar (11) PRIMARY KEY,
        player_name varchar (25) NOT NULL);
```

2.9 LiveMatch

Relational Model:

live_match(livemat_id, data);

Functional Dependencies:

livemat_id -> data

Candidate Keys:

{(livemat_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE live_match(
        livemat_id varchar (11) PRIMARY KEY,
        data);
```

2.10 Community

Relational Model:

community(comm_id, comm_name, news);

Functional Dependencies:

comm_id -> comm_name, news

Candidate Keys:

{(comm_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE community(
        comm_id varchar (11) PRIMARY KEY,
        comm_name varchar (25) NOT NULL,
        news varchar (2000) NOT NULL);
```

2.11 Lottery Ticket

Relational Model:

lottery_ticket(lot_id, money_given, degree, fee, digits);

Functional Dependencies:

lot_id -> money_given, degree, fee, digits

Candidate Keys:

{(lot_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE lottery_ticket(
        lot_id varchar (11) PRIMARY KEY,
        money_given int (12) NOT NULL,
        degree int (12) NOT NULL,
        fee int (12) NOT NULL,
        digits int (6) NOT NULL);
```

2.12 Survey

Relational Model:

survey(survey_id, questions, results);

Functional Dependencies:

survey_id -> questions, results

Candidate Keys:

{(survey_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE survey(
        survey_id varchar (11) PRIMARY KEY,
        question varchar (50) NOT NULL,
        answer varchar (100) NOT NULL);
```

2.13 Post

Relational Model:

post(<u>post_id</u>, date, post_type);

Functional Dependencies:

post_id -> date, post_type

Candidate Keys:

{(post_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE post(
        post_id varchar (11) PRIMARY KEY,
        date date NOT NULL,
        post_type int (1) NOT NULL);
```

2.14 Comment

Relational Model:

comment(comment_id, content, date);

Functional Dependencies:

comment_id -> content, date

Candidate Keys:

{(comment_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE comment(
        comment_id varchar (11) PRIMARY KEY,
        content varchar (500) NOT NULL,
        date date NOT NULL);
```

2.15 Friend

Relational Model:

friend(user_name, since);

Functional Dependencies:

user_name -> since

Candidate Keys:

{(user_name)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE friend(
        user_name varchar (25) NOT NULL,
        since date NOT NULL,
        FOREIGN KEY (user_name) REFERENCES user(username));
```

2.16 Follow

Relational Model:

follow(TCK,ed_id);

Functional Dependencies:

None.

Candidate Keys:

{(TCK),(ed_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE follow(
        TCK varchar (11) NOT NULL,
        ed_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (ed_id) REFERENCES editor(ed_id));
```

2.17 Buy

Relational Model:
buy(TCK, lot_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(lot_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE buy(
        TCK varchar (11) NOT NULL,
        lot_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (lot_id) REFERENCES lottery(lot_id));
```

2.18 Edit_Lott

Relational Model:
edit_lott(lot_id, admin_id);

Functional Dependencies:
None.

Candidate Keys:

{(lot_id),(admin_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE edit_lott(
        lot_id varchar (11) NOT NULL,
        admin_id varchar (11) NOT NULL,
        FOREIGN KEY (lot_id) REFERENCES lottery(lot_id),
        FOREIGN KEY (admin_id) REFERENCES admin(admin_id));
```

2.19 Fill

Relational Model:
fill(survey_id,TCK);

Functional Dependencies:
None.

Candidate Keys:

{(survey_id),(TCK)}

Normal Form: BCNF

Table Definition:

CREATE TABLE fill(

survey_id varchar (11) NOT NULL,

TCK varchar (11) NOT NULL,

FOREIGN KEY (survey_id) REFERENCES survey(survey_id),

FOREIGN KEY (TCK) REFERENCES user(TCK));

2.20 Create_Betslip

Relational Model:
create_betslip(<u>betslip_id</u>, <u>*TCK*</u>);

Functional Dependencies:
None.

Candidate Keys:

{(betslip_id),(TCK)}

Normal Form: BCNF

Table Definition:

CREATE TABLE create_betslip(

betslip_id varchar (11) NOT NULL,

TCK varchar (11) NOT NULL,

FOREIGN KEY (betslip_id) REFERENCES bet_slip(betslip_id),

FOREIGN KEY (TCK) REFERENCES user(TCK));

2.21 Edit_Sur

Relational Model:
edit_sur(survey_id, admin_id);

Functional Dependencies:
None.

Candidate Keys:

{(survey_id),(admin_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE edit_sur(
        survey_id varchar (11) NOT NULL,
        admin_id varchar (11) NOT NULL,
        FOREIGN KEY (survey_id) REFERENCES survey(survey_id),
        FOREIGN KEY (admin_id) REFERENCES admin(admin_id));
```

2.22 Edit

Relational Model:
edit(admin_id, bet_id);

Functional Dependencies:
None.

Candidate Keys:

{(admin_id),(bet_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE edit(
        admin_id varchar (11) NOT NULL,
        bet_id varchar (11) NOT NULL,
        FOREIGN KEY (admin_id) REFERENCES admin(admin_id),
        FOREIGN KEY (bet_id) REFERENCES bet(bet_id));
```

2.23 Have

Relational Model:
have(<u>betslip_id</u>, <u>bet_id</u>);

Functional Dependencies:
None.

Candidate Keys:

{(betslip_id),(bet_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE have(
        betslip_id varchar (11) NOT NULL,
        bet_id varchar (11) NOT NULL,
        FOREIGN KEY (betslip_id) REFERENCES betslip(betslip_id),
        FOREIGN KEY (bet_id) REFERENCES bet(bet_id));
```

2.24 Create

Relational Model:
create(TCK, post_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(post_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE create(
        TCK varchar (11) NOT NULL,
        post_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (post_id) REFERENCES post(post_id));
```

2.25 Like

Relational Model:
like(TCK, post_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(post_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE like(
        TCK varchar (11) NOT NULL,
        post_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (post_id) REFERENCES post(post_id));
```

2.26 Share

Relational Model:
share(TCK, post_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(post_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE share(
        TCK varchar (11) NOT NULL,
        post_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (post_id) REFERENCES post(post_id));
```

2.27 Be_Fan

Relational Model:
be_fan(TCK, comm_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(comm_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE be_fan(
        TCK varchar (11) NOT NULL,
        comm_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (comm_id) REFERENCES community(comm_id));
```

2.28 Follow

Relational Model:
follow(TCK, ed_id);

Functional Dependencies:
None.

Candidate Keys:

{(TCK),(ed_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE follow(
        TCK varchar (11) NOT NULL,
        ed_id varchar (11) NOT NULL,
        FOREIGN KEY (TCK) REFERENCES user(TCK),
        FOREIGN KEY (ed_id) REFERENCES editor(ed_id));
```

2.29 Editor_Betslip

Relational Model:

editor_betslip(edslip_id, balance, valid_or_not);

Functional Dependencies:

edslip_id -> balance, valid_or_not

Candidate Keys:

{(edslip_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor_betslip(
        edslip_id varchar (11) PRIMARY KEY,
        balance int (12),
        valid_or_not bit(1) NOT NULL);
```

2.30 Share_e

Relational Model:
share_e(edslip_id, ed_id);

Functional Dependencies:
None.

Candidate Keys:
{(edslip_id),(ed_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE share_e(
        edslip_id varchar (11) NOT NULL,
        ed_id varchar (11) NOT NULL,
        FOREIGN KEY (edslip_id) REFERENCES editor_betslip(edslip_id),
        FOREIGN KEY (ed_id) REFERENCES editor(ed_id));
```

2.31 Post_Editor

Relational Model:
post_editor(postEdi_id, date, content);

Functional Dependencies:
postEdi_id -> date, content

Candidate Keys:
{(postEdi_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE post_editor(
        postEdi_id varchar (11) PRIMARY KEY,
        date date NOT NULL,
        content varchar (500) NOT NULL);
```

3 User Interface Design and Corresponding SQL statements

3.1 Sign-up Page

Inputs

@user_name, @user_surname, @email, @date_of_birth, @phone_number

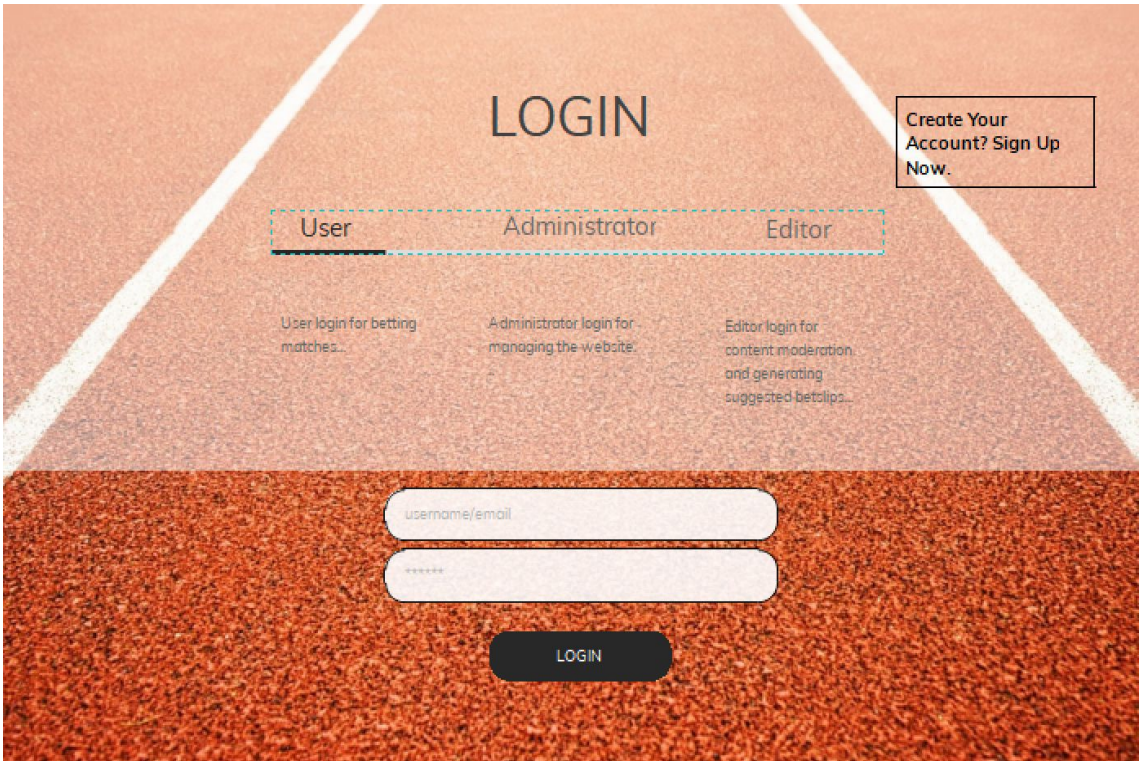
Process

User is directed to this page if the user is a new user, he/she needs to create an account when completed mandatory input fields name, surname, password, username, email, date of birth, phone number. When those fields are filled. The user account will be created and the user will be directed to the main page.

SQL Statement:

INSERT INTO User(name, surname, password, username, email, date of birth, phone number)
VALUES(@name,@surname,@password,@username, @email, @date_of_birth,@phone_number);

3.2 Log-in User & Admin & Editor Page



Inputs

@user_name, @email, @password

Process

The user is directed to this popup when entering the website. This page directs existing users to the main page through username/email and password mandatory input fields. When those fields are entered correctly, the user is directed to the main page. Also if the user is not an existing user, the user should create a new account before being directed to the main page.

SQL Statement:

User Log_in:

```
SELECT user_name
FROM User U
        WHERE @user_name= U.user_name
        AND @password = U.password
        AND @email = U.email;
```

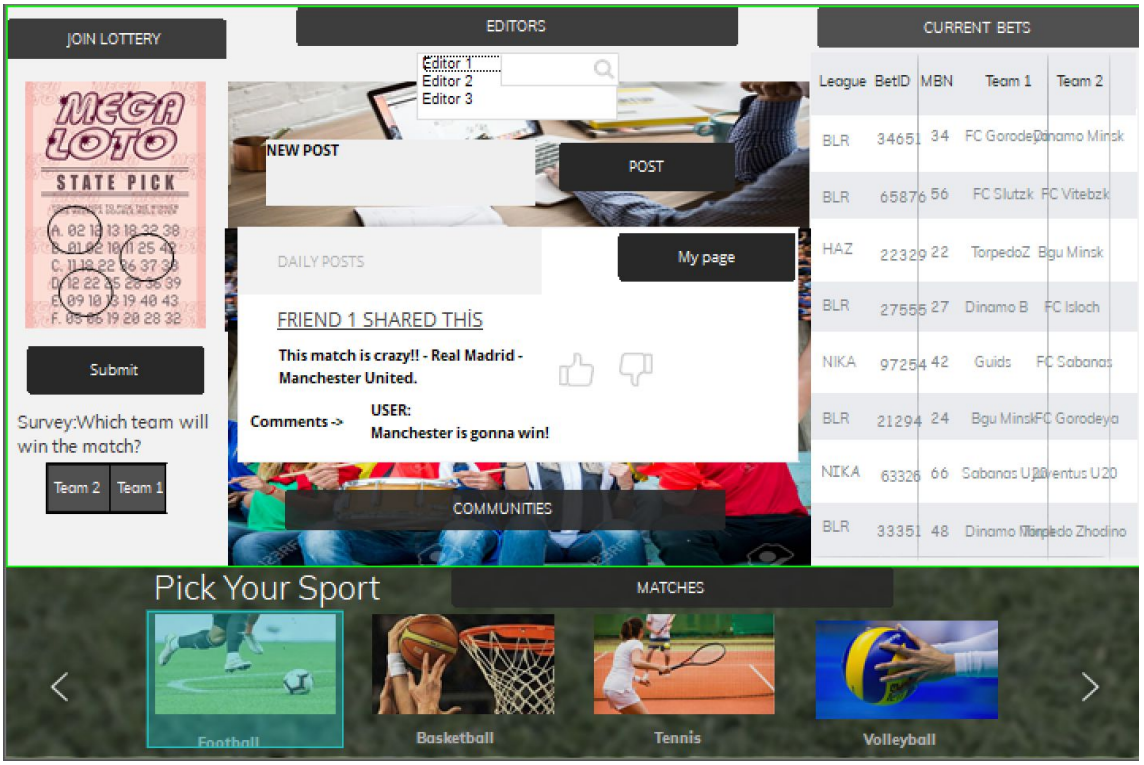
Editor Log_in:

```
SELECT user_name
FROM Editor E
        WHERE @user_name= E.user_name
        AND @password = E.password
        AND @email = E.email;
```

Admin Log_in:

```
SELECT user_name
FROM Admin A
        WHERE @user_name= A.user_name
        AND @password = A.password
        AND @email = A.email;
```

3.3 Main Page for User Page



Process

This is a user main page that shows up after login. The current bets are shown on the right. Users can play lottery tickets and submit them. Also, he/she can take a current survey. To see editors which are followed by the user, User can filter with a search bar at the top. Users can post a new post, comment on friends' activities or posts. Also, he/she can like it. My page button directs the user to his/her personal page. Community button directs the community he/she is a fan of. The matches can be seen with choosing a sport type on the bottom.

SQL Statement:

Comments:

INSERT INTO Comment
VALUES (@comment_id,@text,@date)
    WHERE @text
    NOT IN
        (**SELECT** text
        FROM Comment)

INSERT INTO Comment_list
VALUES (@bet_id,@comment_id)

Pick Your Sport:

SELECT sport_typeName
FROM Matches M
    WHERE @sport_typeName= M.sport_typeName

Post:

INSERT INTO Post
VALUES (@post_id,@text,@date)
    WHERE @text
    NOT IN
        (**SELECT** text
        FROM Post)

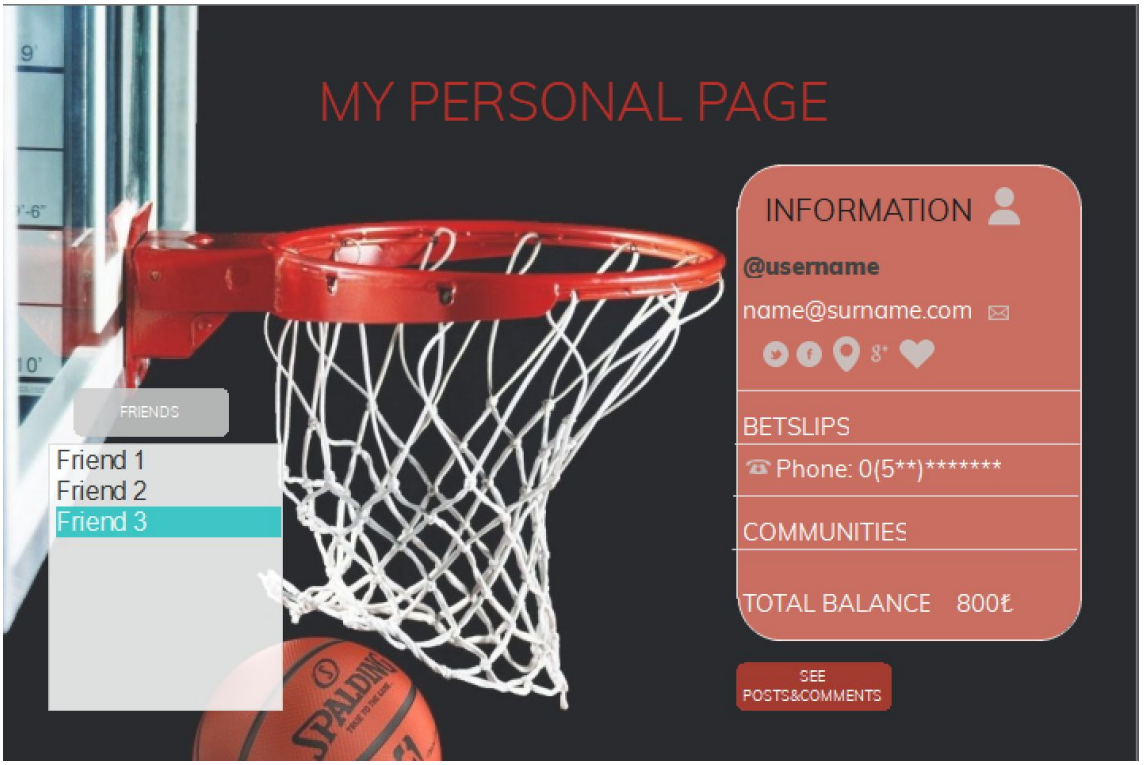
INSERT INTO Post_list
VALUES (@bet_id,@post_id)

Search for Editor:

SELECT editor_id
FROM Editor **NATURAL JOIN** User

WHERE @editor_id = user.id

3.3.1 Personal Page



Process

This page is shown when the User presses the My page button on the main page. The friends can be seen on the left. The see posts and comments button directs User back to the main page. Betslips list page will be directed when clicking to BETSLIPS. Communities button directs the user to the Community page that the user follows. Choosing a friend from the list directs the user to another user's page.

SQL Statement:

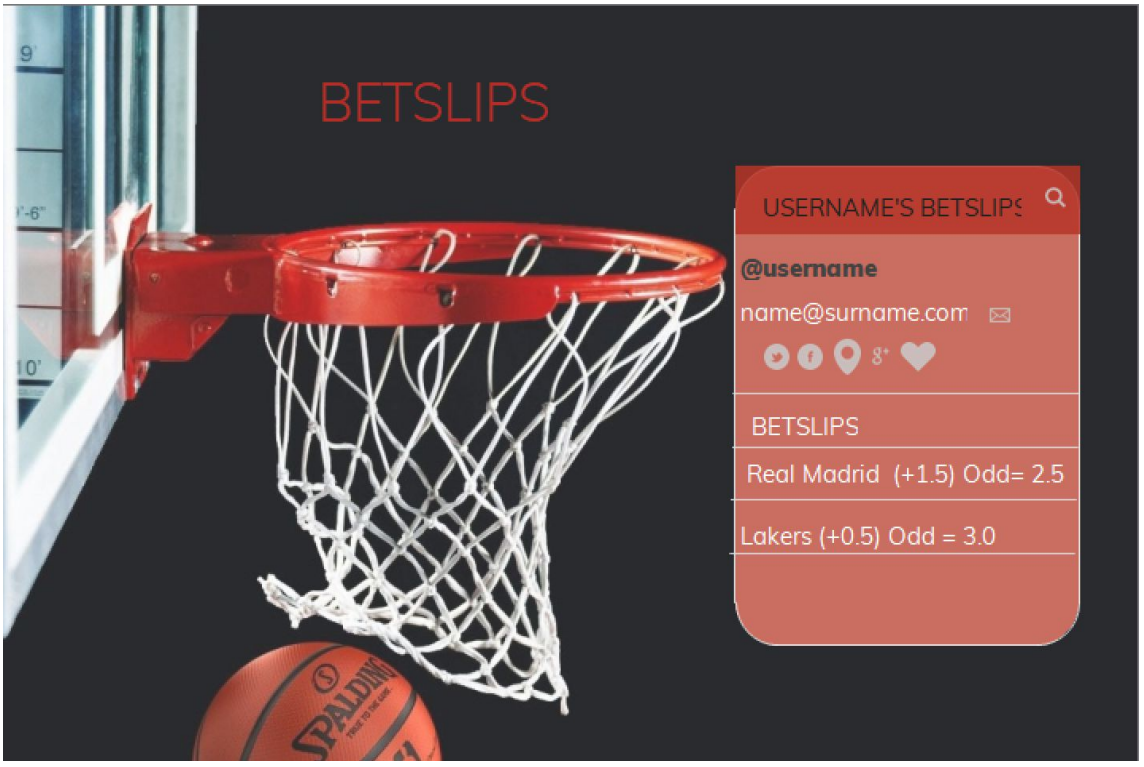
Comments:

```
DELETE FROM Comment
WHERE @comment_id = comment_id
AND comment_id  IN
        (SELECT comment_id
        FROM user_comment
                WHERE @user_id = user_comment.user_id AND @comment_id =
                user_comment.comment_id)

UPDATE Comment
SET text = @text
WHERE @comment_id = comment_id
AND comment_id  IN
        (SELECT comment_id
        FROM user_comment
                WHERE @user_id = user_comment.user_id AND @comment_id =
                user_comment.comment_id)
```

Posts:

```
DELETE FROM Post
WHERE @post_id = post_id
AND post_id  IN
        (SELECT post_id
        FROM user_post
                WHERE @user_id = user_post.user_id AND @post_id = user_post.post_id)

UPDATE Post
SET text = @text
WHERE @post_id = post_id
AND post_id  IN
        (SELECT post_id
        FROM user_post
                WHERE @user_id = user_post.user_id AND @post_id = user_post.post_id)
```

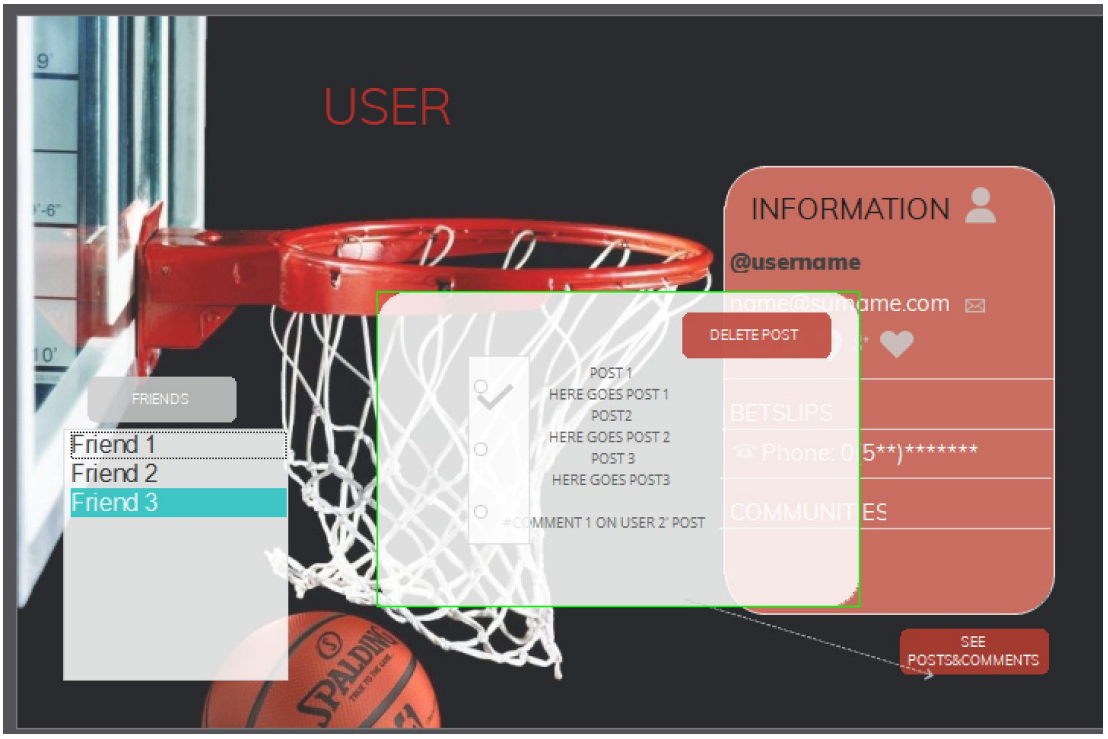
3.3.2 Betslips



Process

This page is directed from clicking to the betslip button on the user's page. it brings the user's betslips currently.

3.3.3 Another User



Process

This is the general page for any user. The current user can add this person as a friend and also check the friends of this user. Also, see the posts and comments on the main page.

SQL Statement:

Comments:

INSERT INTO Comment
VALUES (@comment_id,@text,@date)
    WHERE @text
    NOT IN
        (**SELECT** text
        FROM Comment)

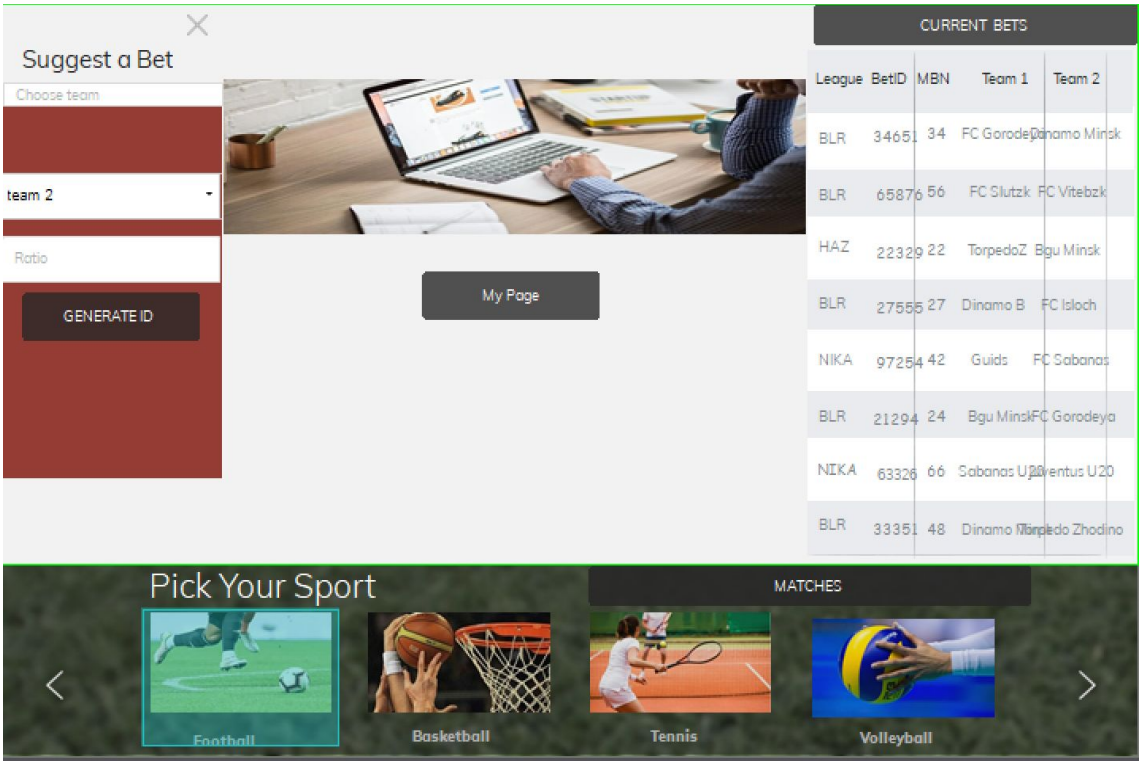
INSERT INTO Comment_list
VALUES (@bet_id,@comment_id)

Post:

INSERT INTO Post
VALUES (@post_id,@text,@date)
    WHERE @text
    NOT IN
        (**SELECT** text
        FROM Post)

INSERT INTO Post_list
VALUES (@bet_id,@post_id)

3.4 Main Page for Editor



SQL Statement:

Pick Your Sport:

SELECT sport_typeName
FROM Matches M
WHERE @sport_typeName= M.sport_typeName

Current Bets:

SELECT match-id, min-bet, league, time, team-name1, team-name2, type, odd-rate from football matches join Team T1, T2

WHERE team-id1 = T1.team-id and team-id2 = T2.team-id

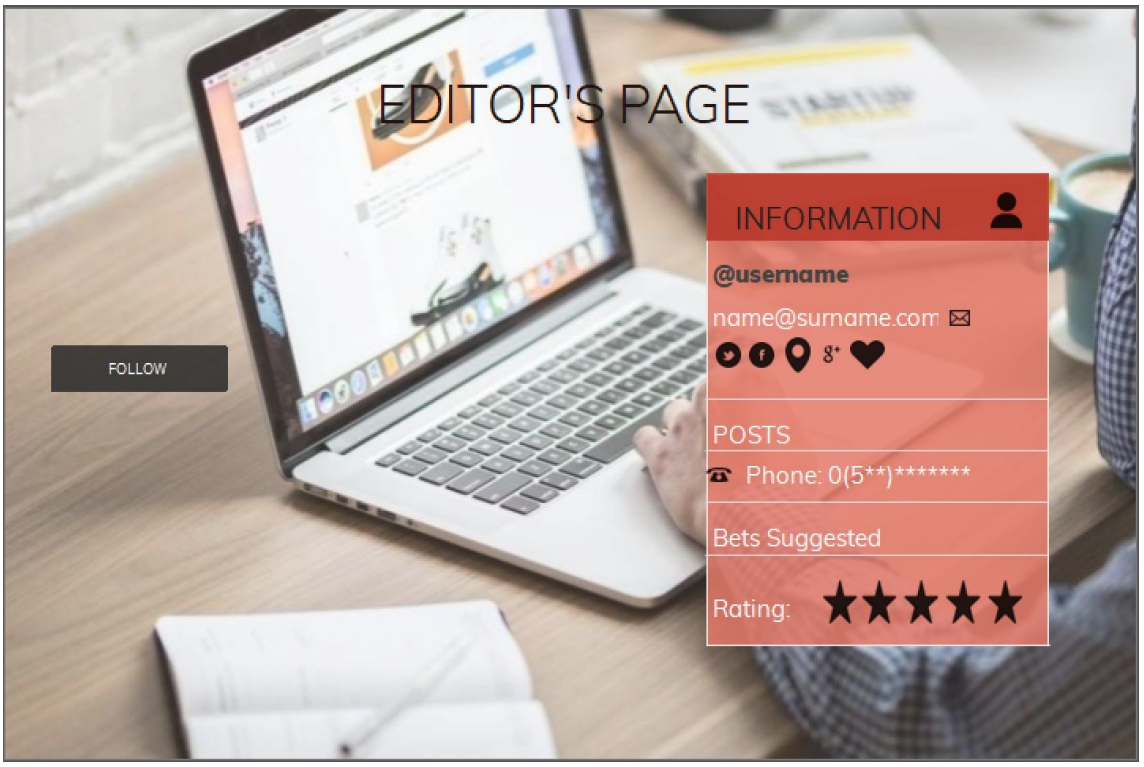
3.4.1 Editor's Page

SQL Statement:

Bets Suggested:

SELECT (*)
FROM Suggest S
WHERE S.bet_id = bet_id

3.4.2 Editor Page to Nonfollowers



SQL Statement:

```
SELECT editor_id
FROM Editor NATURAL JOIN User
        WHERE @user_id = Friend.user_id
UNION
SELECT editor_id
FROM Editor
        WHERE @user_id = editor_id
```

3.4.2.1 Post's of Editor

SQL Statements:

General information and Follow

SELECT editor_id

FROM Editor **NATURAL JOIN** User

    WHERE @user_id = Friend.user_id

UNION

SELECT editor_id

FROM Editor

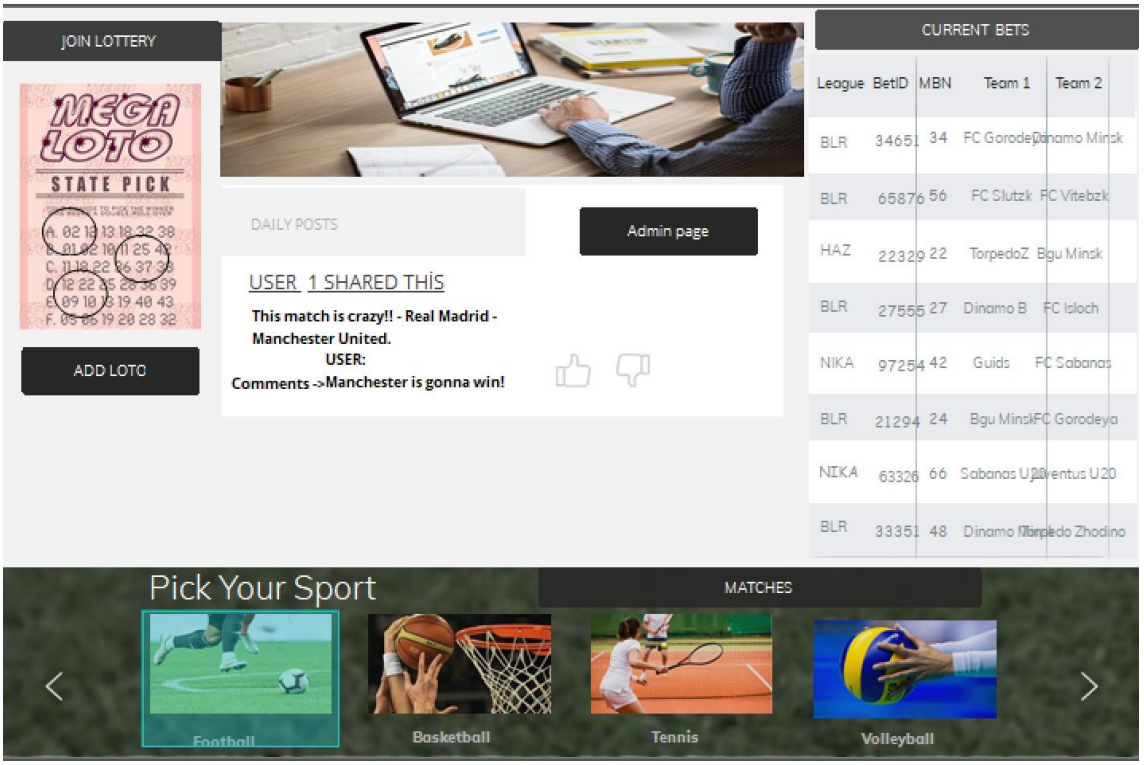
    WHERE @user_id = editor_id

3.4.2 Follower Of Editor

SQL Statement:

SELECT editor_id
FROM Editor **NATURAL JOIN** User
    WHERE @user_id = Friend.user_id
UNION
SELECT editor_id
FROM Editor
    WHERE @user_id = editor_id

3.5 Main Page for Admin



SQL Statement:

Pick Your Sport:

SELECT sport_typeName

FROM Matches M

WHERE @sport_typeName= M.sport_typeName

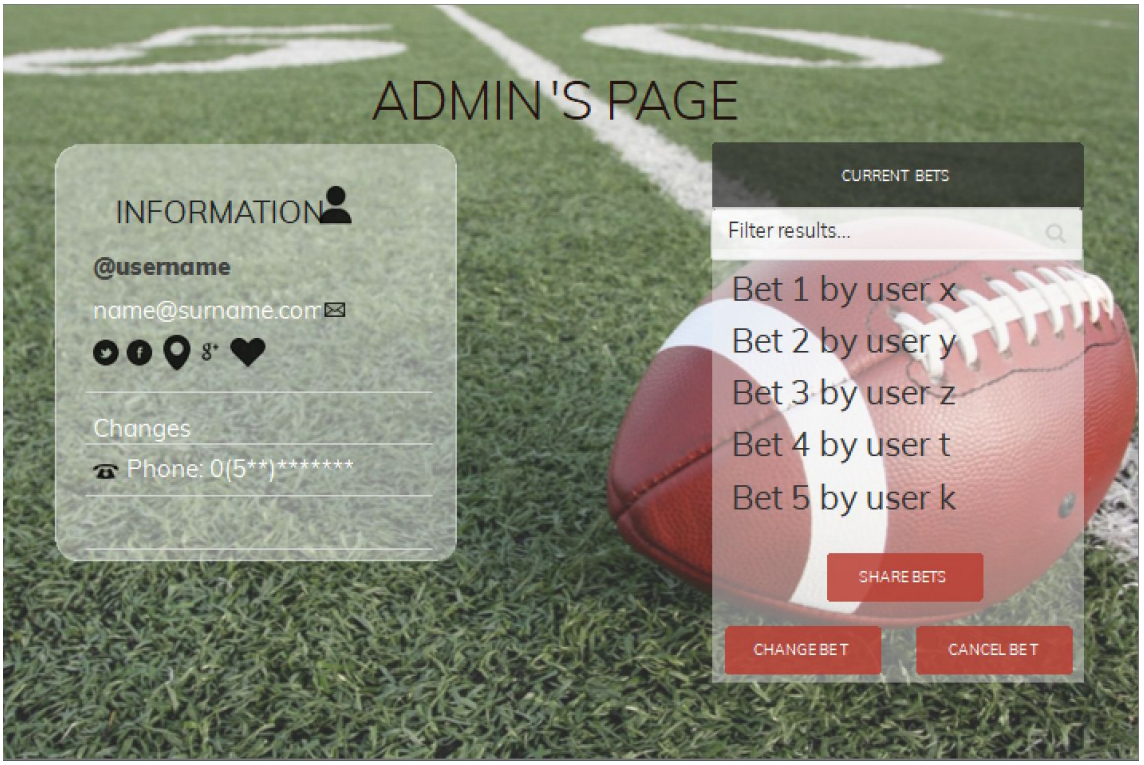
Current Bets:

SELECT match-id, min-bet, league, time, team-name1, team-name2, type, odd-rate

FROM football matches join Team T1, T2

WHERE team-id1 = T1.team-id and team-id2 = T2.team-id

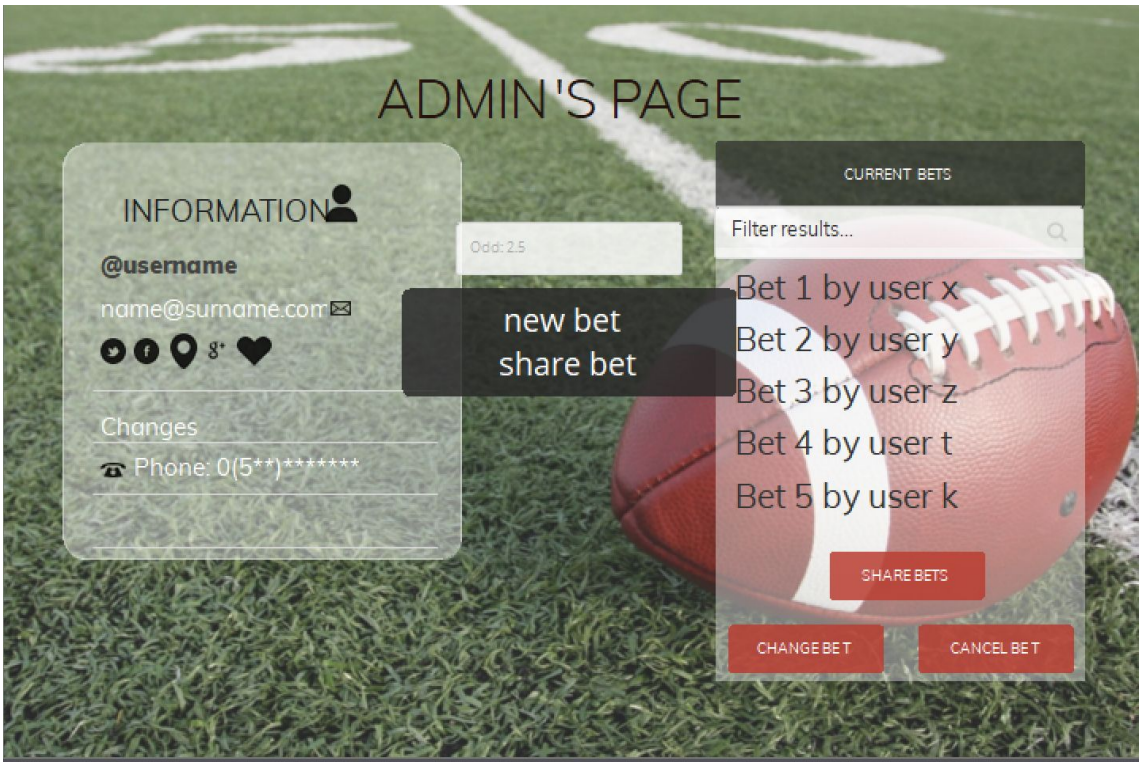
3.5.1 Admin Page



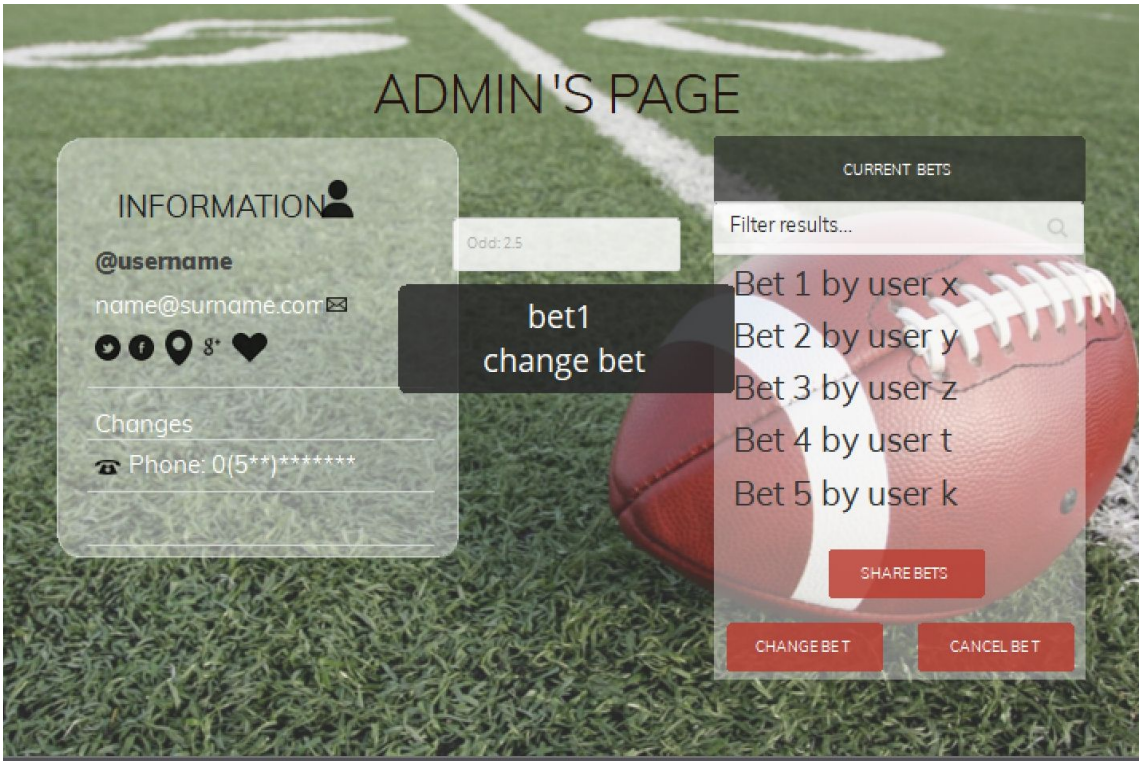
Process

This page gives general information to the Admin of him/herself. Also, it shows the current bets so that the admin can manipulate them. Share bets button make edition pop-up show. Change bet and also directs to the pop up of the editing screen. Cancel bet directly to list and admin chooses which one to cancel.

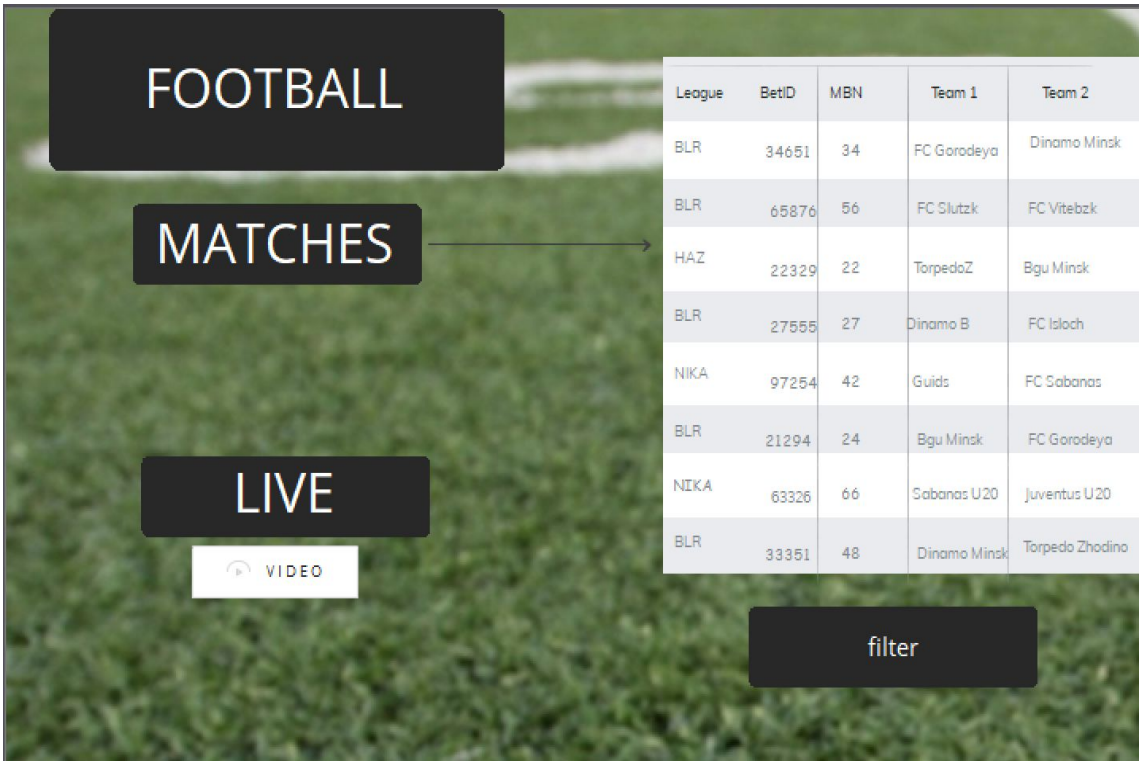
3.5.1.1 Create Bet



3.5.1.2 Change Bet



3.7 Matches



Process

User, Editor or Admin chooses any sport type on their main page. This page is directed from that choice. There is a live match show to watch at the moment. Also, people can filter the matches on the team, mbn, league or any other specifics.

SQL Statements:

Matches:

SELECT name

FROM Match natural join Team_Match natural join Team natural join Player_Team natural join player

WHERE match-id = @match-id

SELECT*

FROM Match natural join contains natural join
odd **WHERE** match-id = @match-id

3.8 Community

Process

From any main page or personal pages, when the user clicks the community button, it directs to the community page, this page also shows the one following currently. The second screen is directed from the button “You follow”.

References

[1] "Tuttur.com Iddaa Spor Toto Milli Piyango TJK At Yarışı" [Online] Available: www.tuttur.com, Accessed March 2, 2020

[2] "Flowchart Maker & Online Diagram Software" [Online] Available: www.draw.io, Accessed March 2, 2020

[3] “Free prototyping tool for web & mobile apps”[Online] Available: www.justinmind.com, Accessed April 7, 2020