CS 353 Database Systems

Project Design Report

Group 1 - Section 3

AlphaBet

Social Betting Platform

Mert Aslan 21702818

Rahmiye Büşra Büyükgebiz 21702019

Ozan Aydın 21702957

Yüce Hasan Kılıç 21704070

1. Revised E/R Model

After our teaching assistant reviewed our proposal report, we received feedback from the assistant and revised the E/R model according to the feedback. We have corrected the mistakes on the E/R model and removed some redundant attributes or tables. The changes are as follows;

1.1. Changes to entities

- We decided that match_id alone should be the primary key for the match entity, thus start_date is no longer part of the primary key.
- For the team entity, team_name no longer is part of a primary key, instead, team_id which is a foreign key to the competitor entity is the primary key by itself.
- We removed the like_count attribute from the comment entity. We now use the relationship comment_likes to count the number of likes of each comment in the system.
- We added an attribute called active to the bet entity. This attribute keeps track of the active bet, which is the latest bet that has been altered by the admin. This attribute is then used to display the most up-to-date bets in the homepage. Bets which have already been placed by users and marked as inactive by admins still remain and are used to reward the users if the bet is successful.
- We added an attribute called result to the bet entity. This attribute keeps track of the status of the bets. The possible values for this attribute are “lost”, “pending”, or “won”.
- We added third, fourth and fifth set winner attributes to the tennis_result entity, as instructed by the feedback that we received.
- We changed the names of our entities from plural form to singular form, as instructed by the feedback that we received.

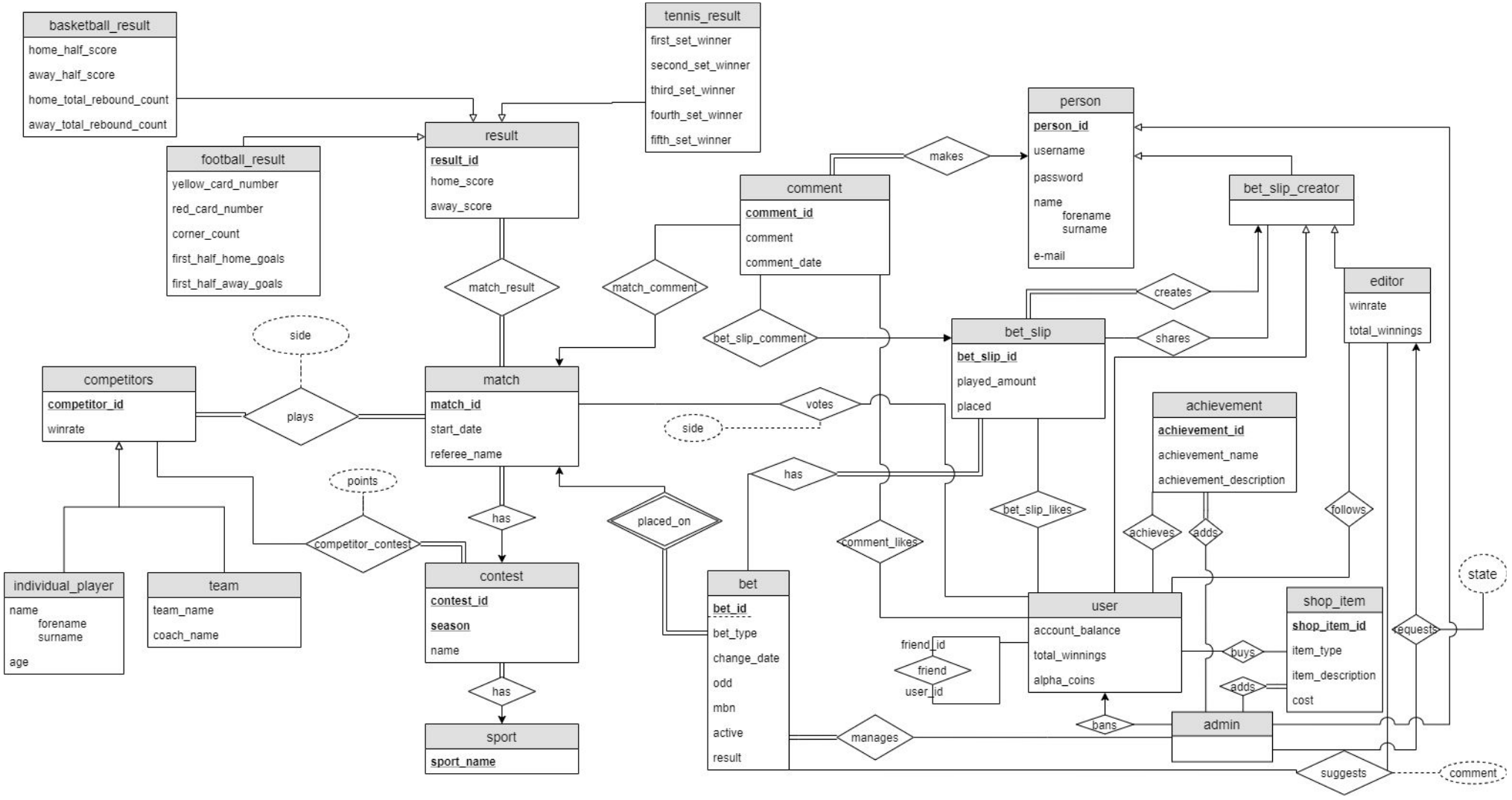
1.2.Changes to relationships

- We added a relationship between user and comment entities called likes, to keep track of the likes given to each comment in the system.
- We added a relationship between user and bet_slip entities called bet_slip_likes, to keep track of the likes given to each bet slip in the system.
- We changed the name of the banned_users relationship to bans.
- We removed the relationship between user and bet_slip entities, as this relationship is already captured using the creates relationship between bet_slip_creator and bet_slip entities.
- We added a new relationship called suggests between editor and bet entities. This relationship keeps track of the suggestions made by editors on a single bet. This relationship has an attribute which is a comment made while suggesting the bet.

- We added a new relationship called shares between bet_slip_creator and bet_slip entities. This keeps track of the shared bet slips, to be shown in the feed section of the system.
- We added a new relationship called requests between admin and editor entities. This keeps track of the requests to be an editor made in the login stage by editor candidates. Admins can view these requests from their dashboard, and accept or decline them.
- We added all of the missing total participations between relationships. These are:
    - For both result and match entities between “match_result” relationship
    - For both competitor and match between “competitor_match” relationship.
    - For bet_slip entity in the “placed_bets” relationship.
    - For contest entity in the “competitor_contest” relationship.
    - For comment entity in the “makes” relationship
    - For bet_slip entity in the “has” relationship.
    - For contest entity in the “has” relationship.

1.3.New Features

- We removed the group entity from the database. Our project no longer involves groups. Instead, we have added two new features which are a market and an achievement system. By betting, users now gain alpha coins, which they can trade in for items in the market. Also, by satisfying certain requirements, users now gain achievements, which show up in their profile page.
- In order to implement the market system, we added a new entity called shop_item. This entity keeps track of the items that are sold in the market.
- In order to implement the achievement system, we added a new entity called achievement. This entity keeps track of the achievements that can be achieved by users in the system. The status of achievements for each user will be detected using triggers.
- We added a new relationship called adds between admin and shop_item. This keeps track of items in the market and the admin making the insertion.
- We added a new relationship called buys between user and shop_item. This keeps track of items bought by users.
- We added a new relationship called adds between admin and achievement. This keeps track of achievements in the system and the admin making the insertion.
- We added a new relationship called achieves between user and achievement. This keeps track of achievements gained by users.


basketball_result
home_half_score
away_half_score
home_total_rebound_count
away_total_rebound_count
tennis_result
first_set_winner
second_set_winner
third_set_winner
fourth_set_winner
fifth_set_winner
football_result
yellow_card_number
red_card_number
corner_count
first_half_home_goals
first_half_away_goals
result
result_id
home_score
away_score
match_result
match_comment
comment
comment_id
comment
comment_date
makes
person
person_id
username
password
name
forename
surname
e-mail
bet_slip_creator
creates
editor
winrate
total_winnings
bet_slip_comment
bet_slip
bet_slip_id
played_amount
placed
shares
side
competitors
competitor_id
winrate
plays
match
match_id
start_date
referee_name
votes
side
achievement
achievement_id
achievement_name
achievement_description
points
has
placed_on
has
bet_slip_likes
comment_likes
achieves
adds
follows
competitor_contest
individual_player
name
forename
surname
age
team
team_name
coach_name
contest
contest_id
season
name
has
sport
sport_name
bet
bet_id
bet_type
change_date
odd
mbn
active
result
manages
friend_id
friend
user_id
user
account_balance
total_winnings
alpha_coins
buys
adds
bans
admin
shop_item
shop_item_id
item_type
item_description
cost
equests
state
suggests
comment

2. Relation Schemas

2.1.Person

Relational Model: person(<u>person_id,</u> username, password, forename, surname, e-mail)

Functional Dependencies:
person_id -> username, password, forename, surname, e-mail
username -> person_id
e-mail -> person_id

Candidate Keys: { (person_id), (username), (e-mail) }

Normal Form: BCNF

Table Definition:
```
CREATE TABLE person(
    person_id INT,
    username VARCHAR(16) NOT NULL UNIQUE,
    password VARCHAR(16) NOT NULL,
    forename VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    PRIMARY KEY(person_id)
    );
```

2.2.User

Relational Model: user(<u>user_id</u>, account_balance, total_winnings, alpha_coins)
user_id: FK to bet_slip_creator(creator_id)

Functional Dependencies: user_id -> account_balance, total_winnings, alpha_coins

Candidate Keys: { (user_id) }

Normal Form: BCNF

Table Definition:
```
CREATE TABLE user(
    user_id INT,
    account_balance INT,
    total_winnings INT,
    alpha_coins INT,
    PRIMARY KEY(user_id),
```

```
FOREIGN KEY (user_id) REFERENCES
bet_slip_creator(creator_id)
ON DELETE CASCADE
);
```

2.3.User Friend

Relational Model: user_friend(<u>user_id, friend_id</u>)
user_id: FK to user(user_id)
friend_id: FK to user(user_id)

Functional Dependencies: user_id, friend_id -> user_id, friend_id

Candidate Keys: {(user_id, friend_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE user_friend(

```
user_id INT,
friend_id INT,
PRIMARY KEY(user_id, friend_id),
FOREIGN KEY(user_id) REFERENCES user(user_id) ON
DELETE CASCADE,
FOREIGN KEY(friend_id) REFERENCES user(user_id)
ON DELETE CASCADE
);
```

2.4.Editor

Relational Model: editor(<u>editor_id</u>, winrate, total_winnings)
editor_id: FK to bet_slip_creator(creator_id)

Functional Dependencies: editor_id -> winrate, total_winnings

Candidate Keys: { (editor_id) }

Normal Form: BCNF

Table Definition: CREATE TABLE editor (

```
editor_id INT,
winrate INT,
total_winnings INT,
PRIMARY KEY(editor_id),
FOREIGN KEY(editor_id) REFERENCES
ON DELETE CASCADE
```

```
bet_slip_creator(creator_id)
);
```

2.5.User Follows

Relational Model: user_follows(<u>editor_id, user_id</u>)
editor_id: FK to editor(editor_id)
user_id: FK to user(user_id)

Functional Dependencies: editor_id, user_id -> editor_id, user_id

Candidate Keys: { (editor_id, user_id) }

Normal Form: BCNF

Table Definition:
```
CREATE TABLE user_follows(
editor_id INT,
user_id INT,
PRIMARY KEY(editor_id, user_id),
FOREIGN KEY (editor_id) REFERENCES editor(editor_id)
ON DELETE CASCADE,
FOREIGN KEY (user_id) REFERENCES user(user_id)
ON DELETE CASCADE
);
```

2.6.Admin

Relational Model: admin(<u>admin_id</u>)
admin_id: FK to person(person_id)

Functional Dependencies: admin_id -> admin_id (Trivial dependency)

Candidate Keys: { (admin_id) }

Normal Form: BCNF

Table Definition:
```
CREATE TABLE admin (
admin_id INT,
PRIMARY KEY(admin_id)
FOREIGN KEY (admin_id) REFERENCES person(person_id)
ON DELETE CASCADE
);
```

2.7.Bet

Relational Model:
bet(bet_id, match_id, bet_type, change_date, odd, mbn, result, active)
match_id FK to match(match_id)

Functional Dependencies:
bet_id, match_id -> bet_type, change_date, odd, mbn, result, active

Candidate Keys: {(bet_id, match_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bet (
    bet_id INT,
    match_id INT,
    bet_type VARCHAR(30),
    change_date TIMESTAMP,
    odd FLOAT(5, 2),
    mbn INT,
    result VARCHAR(8),
    active BOOLEAN,
    PRIMARY KEY(bet_id, match_id),
    FOREIGN KEY (match_id) REFERENCES match(match_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK result IN ('WON', 'RESULT', 'PENDING')
    );
```

2.8.Bet Slip

Relational Model: bet_slip(bet_slip_id, creator_id, placed, played_amount)
creator_id: FK to bet_slip_creator(creator_id)

Functional Dependencies: bet_slip_id -> creator_id, placed, played_amount

Candidate Keys: {(bet_slip_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bet_slip (
    bet_slip_id INT,
    placed BOOL,
    played_amount INT,
    PRIMARY KEY(bet_slip_id),
    );
```

2.9.Shared Bet Slip

Relational Model: shared_bet_slip(bet_slip_id, sharer_id)

Functional Dependencies: bet_slip_id, sharer_id -> bet_slip_id, sharer_id

Candidate Keys: {(bet_slip_id, sharer_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE shared_bet_slip(
bet_slip_id INT,
sharer_id INT,
PRIMARY KEY(bet_slip_id, sharer_id)
FOREIGN KEY(bet_slip_id) REFERENCES
bet_slip(bet_slip_id),
FOREIGN KEY(sharer_id) REFERENCES
bet_slip_creator(creator_id) ON DELETE CASCADE
)
```

2.10.Suggested Bet

Relational Model: suggested_bet(editor_id, bet_id, match_id, comment)
editor_id: FK to editor(editor_id)
bet_id, match_id: FK to bet_id(bet_id, match_id)

Functional Dependencies: editor_id, bet_id, match_id -> comment

Candidate Keys: {(editor_id, bet_id, match_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE suggested_bet(
editor_id INT,
bet_id INT,
match_id INT
comment VARCHAR(255),
PRIMARY KEY(editor_id, bet_id),
FOREIGN KEY(editor_id) REFERENCES editor(editor_id) ON
DELETE CASCADE,
FOREIGN KEY(bet_id, match_id) REFERENCES bet(bet_id,
match_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

2.11.Bet Slip Like

Relational Model: bet_slip_like(<u>user_id, bet_slip_id</u>)
user_id: FK to user(user_id)
bet_slip_id: FK to bet_slip(bet_slip_id)

Functional Dependencies: user_id, bet_slip_id -> user_id, bet_slip_id

Candidate Keys: {(bet_slip_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bet_slip_like(
    user_id INT,
    bet_slip_id INT,
    PRIMARY KEY(user_id, bet_slip_id),
    FOREIGN KEY(user_id) REFERENCES user(user_id) ON
    DELETE CASCADE,
    FOREIGN KEY(bet_slip_id) REFERENCES
    bet_slip(bet_slip_id) ON DELETE CASCADE ON UPDATE
    CASCADE
    );
```

2.12.Included Bet

Relational Model: included_bet(<u>bet_slip_id, match_id</u>, bet_id)
bet_slip_id: FK to bet_slip(bet_slip_id)

bet_id, match_id: FK to bet(bet_id, match_id)

match_id: FK to match(match_id)

Functional Dependencies: bet_slip_id, match_id -> bet_id

Candidate Keys: {(bet_slip_id, match_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE included_bet(
    bet_slip_id INT,
    bet_id INT,
    match_id INT,
    PRIMARY KEY(bet_slip_id, match_id),
```

```
FOREIGN KEY(bet_slip_id) REFERENCES
bet_slip(bet_slip_id) ON DELETE CASCADE ON UPDATE
CASCADE
FOREIGN KEY(bet_id, match_id) REFERENCES bet(bet_id,
 match_id) ON DELETE CASCADE ON UPDATE CASCADE
FOREIGN KEY(match_id) REFERENCES match(match_id)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

2.13.Bet Slip Creator

Relational Model: bet_slip_creator(creator_id)
creator_id: FK to person(person_id)

Functional Dependencies: creator_id -> creator_id

Candidate Keys: {(creator_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bet_slip_creator (
creator_id INT,
PRIMARY KEY(creator_id),
FOREIGN KEY(creator_id) REFERENCES person(person_id)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

2.14.Sport

Relational Model: sport(sport_name)

Functional Dependencies: sport_name -> sport_name

Candidate Keys: {(sport_name)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE sport(
sport_name VARCHAR(20),
PRIMARY KEY(sport_name)
)
```

2.15.Contest

Relational Model: contest(contest_id, sport_name, season, name)

sport_name: FK to sport(sport_name)

Functional Dependencies: contest_id, sport_name, season -> name

Candidate Keys: {(contest_id, sport_name, season)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE contest(
contest_id INT,
sport_name VARCHAR(20),
season VARCHAR(9),
name VARCHAR(30),
PRIMARY KEY(contest_id, season),
FOREIGN KEY(sport_name) REFERENCES
sport(sport_name) ON DELETE CASCADE ON UPDATE
CASCADE,
CHECK(sport_name IN('TENNIS, 'FOOTBALL',
'BASKETBALL'))
);
```

2.16.Match

Relational Model: match(<u>match_id</u>, start_date, contest_id, season, sport_name referee_name)

contest_id, sport_name, season: FK to contest(contest_id, sport_name, season)
sport_name: FK to sport(sport_name)

Functional Dependencies:
match_id -> start_date, contest_id, season_id, sport_name,referee_name
start_date, referee_name -> match_id
Candidate Keys: {(match_id), (start_date, referee_name)}

Normal Form: 3NF

Table Definition:
```
CREATE TABLE match(
match_id INT,
start_date TIMESTAMP,
contest_id INT,
season VARCHAR(9),
sport_name VARCHAR(15),
referee_name VARCHAR(40) ,
PRIMARY KEY(match_id),
FOREIGN KEY(contest_id, season) REFERENCES
contest(contest_id, season) ON DELETE CASCADE ON
UPDATE CASCADE,
```

```
FOREIGN KEY(sport_name) REFERENCES
sport(sport_name) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

2.17.Competitor

Relational Model: competitor(<u>competitor_id</u>, winrate)

Functional Dependencies: competitor_id -> winrate

Candidate Keys: {(competitor_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE competitor(

```
competitor_id INT,
winrate FLOAT,
PRIMARY KEY(competitor_id)
);
```

‘

2.18.Competitor Match

Relational Model: competitor_match(<u>competitor_id, match_id</u>, side)
competitor_id: FK to competitor(competitor_id)
match_id: FK to match(match_id)

Functional Dependencies: competitor_id, match_id -> side

Candidate Keys: {(competitor_id, match_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE competitor_match(

```
competitor_id INT,
match_id INT,
side VARCHAR(4),
PRIMARY KEY (competitor_id, match_id),
FOREIGN KEY (competitor_id) REFERENCES
competitor(competitor_id) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY (match_id) REFERENCES match(match_id)
ON DELETE CASCADE ON UPDATE CASCADE,
CHECK(side IN (‘HOME’, ‘AWAY’))
);
```

2.19.Competitor Contest

Relational Model: competitor_contest(<u>competitor_id, contest_id, season</u>, points)
competitor_id: FK to competitor(competitor_id)
contest_id, season: FK to contest(contest_id, season)

Functional Dependencies: competitor_id, contest_id, season -> points

Candidate Keys: {(competitor_id, contest_id, season)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE competitor_contest(
    competitor_id INT,
    contest_id INT,
    season VARCHAR(9),
    points INT,
    PRIMARY KEY(competitor_id, contest_id, season),
    FOREIGN KEY (competitor_id) REFERENCES
    competitor(competitor_id) ON DELETE CASCADE ON
    UPDATE CASCADE,
    FOREIGN KEY(contest_id, season) REFERENCES
    contest(contest_id, season) ON DELETE CASCADE ON
    UPDATE CASCADE
    );
```

2.20.Individual Player

Relational Model: individual_player(<u>player_id</u>, forename, surname, age)
player_id: FK to competitor(competitor_id)

Functional Dependencies: player_id -> forename, surname, age

Candidate Keys: {(player_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE individual_player (
    player_id INT,
    forename VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    age INT,
    PRIMARY KEY(player_id),
```

```
FOREIGN KEY(player_id) REFERENCES
competitor(competitor_id) ON DELETE CASCADE ON
UPDATE CASCADE
)
```

2.21.Team

Relational Model: team(<u>team_id</u>, team_name, coach_name)
team_id: FK to competitor(competitor_id)

Functional Dependencies: team_id -> team_name, coach_name

Candidate Keys: {(team_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE team (

```
team_id INT,
team_name VARCHAR(30) NOT NULL,
coach_name VARCHAR(40),
PRIMARY KEY(team_id),
FOREIGN KEY(team_id) REFERENCES
competitor(competitor_id) ON DELETE CASCADE ON
UPDATE CASCADE
)
```

2.22.Result

Relational Model: result(<u>result_id, match_id</u>, home_score, away_score)
match_id: FK to match(match_id)

Functional Dependencies: result_id, match_id -> home_score, away_score

Candidate Keys: {(result_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE result (

```
result_id INT,
home_score INT,
away_score INT,
match_id INT,
PRIMARY KEY(result_id, match_id)
```

```
FOREIGN KEY(match) REFERENCES match(match_id) ON
DELETE CASCADE ON UPDATE CASCADE
);
```

2.23.Basketball Result

Relational Model: basketball_result(result_id, home_half_score, away_half_score, home_total_rebound_count, away_total_rebound_count)
result_id: FK to result(result_id)

Functional Dependencies: result_id -> home_half_score, away_half_score, home_total_rebound_count, away_total_rebound_count

Candidate Keys: {(result_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE basketball_result (
result_id INT,
home_half_score INT,
away_half_score INT,
home_total_rebound_count INT,
away_total_rebound_count INT,
PRIMARY KEY (result_id),
FOREIGN KEY (result_id) REFERENCES result(result_id) ON
DELETE CASCADE ON UPDATE CASCADE
);
```

2.24.Football Result

Relational Model: football_result(result_id, yellow_card_number, red_card_number, corner_count, first_half_home_goals, first_half_away_goals)
result_id: FK to result(result_id)

Functional Dependencies: result_id -> yellow_card_number, red_card_number, corner_count, first_half_home_goals, first_half_away_goals

Candidate Keys: {(result_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE football_result(
result_id INT,
yellow_card_number INT,
red_card_number INT,
```

```
corner_count INT,
first_half_home_goals INT,
first_half_away_goals INT,
PRIMARY KEY(result_id),
FOREIGN KEY(result_id) REFERENCES result(result_id) ON
DELETE CASCADE ON UPDATE CASCADE
);
```

2.25.Tennis Result

Relational Model: tennis_result(result_id, first_set_winner, second_set_winner)
result_id: FK to result(result_id)

Functional Dependencies: result_id -> first_set_winner, second_set_winner

Candidate Keys: {(result_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE tennis_result(

```
result_id INT,
first_set_winner VARCHAR(4),
second_set_winner VARCHAR(4),
PRIMARY KEY(result_id),
FOREIGN KEY(result_id) REFERENCES result(result_id) ON
DELETE CASCADE ON UPDATE CASCADE,
CHECK(first_set_winner IN ('HOME', 'AWAY')),
CHECK(result_set_winner IN ('HOME', 'AWAY'))
);
```

2.26.Comment

Relational Model: comment(comment_id, person_id, comment, comment_date)
person_id: FK to person(person_id)

Functional Dependencies: comment_id -> person_id, comment, comment_date

Candidate Keys: {(comment_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE comment(

```
comment_id INT,
person_id INT,
comment VARCHAR(MAX),
comment_date TIMESTAMP,
```

```
PRIMARY KEY(comment_id),
FOREIGN KEY(person_id) REFERENCES person(person_id)
ON DELETE CASCADE ON UPDATE CASCADE

);
```

2.27.Bet Slip Comment

Relational Model: bet_slip_comment(comment_id, bet_slip_id)
comment_id: FK to comment(comment_id)
bet_slip_id: FK to bet_slip(bet_slip_id)

Functional Dependencies: comment_id, bet_slip_id -> comment_id, bet_slip_id

Candidate Keys: {(comment_id, bet_slip_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE bet_slip_comment(

```
comment_id INT,
bet_slip_id INT,
PRIMARY KEY(comment_id, bet_slip_id),
FOREIGN KEY(comment_id) REFERENCES
comment(comment_id) ON DELETE CASCADE ON UPDATE
CASCADE,
FOREIGN KEY(bet_slip_id) REFERENCES
bet_slip(bet_slip_id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

2.28.Match Comment

Relational Model: match_comment(match_id, comment_id)
match_id: FK to match(match_id)
comment_id: FK to comment(comment_id)

Functional Dependencies: match_id, comment_id -> match_id, comment_id

Candidate Keys: {(match_id, comment_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE match_comment(

```
match_id INT,
comment_id INT,
PRIMARY KEY(match_id, comment_id),
```

```
FOREIGN KEY(match_id) REFERENCES match(match_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(comment_id) REFERENCES
comment(comment_id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

2.29.Comment Likes

Relational Model: comment_likes(comment_id, user_id)
comment_id: FK to comment(comment_id)
user_id: FK to user(user_id)

Functional Dependencies: comment_id, user_id -> comment_id, user_id

Candidate Keys: {(comment_id, user_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE comment_likes(

```
comment_id INT,
user_id INT,
PRIMARY KEY(comment_id, user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id) ON
DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(comment_id) REFERENCES
comment(comment_id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

2.30.Votes

Relational Model: votes(user_id, match_id, side)
user_id: FK to user(user_id)
match_id: FK to match(match_id)

Functional Dependencies: user_id, match_id -> side

Candidate Keys: {(user_id, match_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE votes(

```
user_id INT,
match_id INT,
side VARCHAR(8),
```

```
PRIMARY KEY(user_id, match_id),
FOREIGN KEY(user_id) REFERENCES user(user_id) ON
DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(match_id) REFERENCES match(match_id)
ON DELETE CASCADE ON UPDATE CASCADE,
CHECK(side IN ('HOME', 'AWAY'))
);
```

2.31.Manages

Relational Model: manages(<u>admin_id, bet_id, match_id</u>)
admin_id: FK to admin(admin_id)
bet_id, match_id: FK to bet(bet_id, match_id)

Functional Dependencies: admin_id, bet_id, match_id -> admin_id, bet_id, match_id

Candidate Keys: {(admin_id, bet_id, match_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE manages(

```
admin_id INT,
bet_id INT,
match_id INT,
PRIMARY KEY(admin_id, bet_id, match_id),
FOREIGN KEY(admin_id) REFERENCES admin(admin_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(bet_id, match_id) REFERENCES bet(bet_id,
match_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

2.32.Editor Request

Relational Model: editor_request(<u>editor_id,</u> admin_id, state)
admin_id: FK to admin(admin_id)
editor_id: FK to editor(editor_id)

Functional Dependencies: editor_id -> admin_id

Candidate Keys: {(editor_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE approves(

```
editor_id INT
```

```
admin_id INT,
state VARCHAR(20),
PRIMARY KEY(editor_id),
FOREIGN KEY(admin_id) REFERENCES admin(admin_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(editor_id) REFERENCES editor(editor_id) ON
DELETE CASCADE ON UPDATE CASCADE,
CHECK(state IN ('APPROVED', 'PENDING')
);
```

2.33.Shop Item

Relational Model: shop_item(shop_item_id, item_type, item_description, cost)

Functional Dependencies: shop_item_id, item_type-> item_description, cost

Candidate Keys: {(shop_item_id, item_type)}

Normal Form: BCNF

Table Definition: CREATE TABLE shop_item(

```
shop_item INT,
item_type VARCHAR(50),
item_description MEDIUMTEXT,
cost INT,
PRIMARY KEY(shop_item_id, item_type),
);
```

2.34.Bought Item

Relational Model: bought_item(shop_item_id, user_id, item_type)
shop_item_id, item_type: FK to shop_item(shop_item_id, item_type)
user_id: FK to user(user_id)

Functional Dependencies: shop_item_id, user_id, item_type -> shop_item_id, user_id, item_type

Candidate Keys: {(shop_item_id, user_id, item_type)}

Normal Form: BCNF

Table Definition: CREATE TABLE bought_item(

```
shop_item_id INT,
user_id INT,
item_type VARCHAR(50),
```

PRIMARY KEY(shop_item_id, user_id, item_type),
FOREIGN KEY(shop_item_id, item_type) REFERENCES shop_item(shop_item_id, item_type) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(user_id) REFERENCES user(user_id) ON DELETE CASCADE ON UPDATE CASCADE
);

2.35. Added Item

Relational Model: added_item(shop_item_id, item_type, admin_id)

Functional Dependencies: shop_item_id, admin_id, item_type -> shop_item_id, admin_id, item_type

Candidate Keys: {(shop_item_id, admin_id, item_type)}

Normal Form: BCNF

Table Definition: CREATE TABLE added_item(
shop_item_id INT,
item_type VARCHAR(50),
admin_id INT,
PRIMARY KEY (shop_item_id, admin_id, item_type),
FOREIGN KEY(shop_item_id, item_type) REFERENCES shop_item(shop_item_id, item_type) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(admin_id) REFERENCES user(admin_id) ON DELETE CASCADE ON UPDATE CASCADE
);

2.36. Achievement

Relational Model: achievement(achievement_id, achievement_name, achievement_description)

Functional Dependencies: achievement_id -> achievement_name, achievement_description

achievement_name -> achievement_id, achievement_description

Candidate Keys: {(achievement_id), (achievement_name)}

Normal Form: 3NF

Table Definition: CREATE TABLE achievement(

```
achievement_id INT,
achievement_name VARCHAR(40),
achievement_description MEDIUMTEXT,
PRIMARY KEY (achievement_id)
);
```

2.37. Gained Achievement

Relational Model: gained_achievement(<u>achievement_id, user_id</u>)

Functional Dependencies: achievement_id -> user_id

Candidate Keys: {(achievement_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE gained_achievement(

```
achievement_id INT,
user_id INT,
PRIMARY KEY(achievement_id, user_id),
FOREIGN KEY(achievement_id) REFERENCES
achievement(achievement_id) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(user_id) REFERENCES user(user_id) ON
DELETE CASCADE ON UPDATE CASCADE
);
```

2.38. Added Achievement

Relational Model: added_achievement(<u>admin_id, achievement_id</u>)

Functional Dependencies: admin_id, achievement_id -> admin_id, achievement_id

Candidate Keys: {(admin_id, achievement_id)}

Normal Form: BCNF

Table Definition: CREATE TABLE added_achievement(

```
admin_id INT,
achievement INT,
PRIMARY KEY(admin_id, achievement_id),
FOREIGN KEY(admin_id) REFERENCES admin(admin_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(achievement_id) REFERENCES
achievement(achievement_id) ON DELETE CASCADE ON
UPDATE CASCADE,
```

```
);
```

2.39. Bans

Relational Model: bans(<u>user_id,</u> admin_id)

Functional Dependencies: user_id -> admin_id

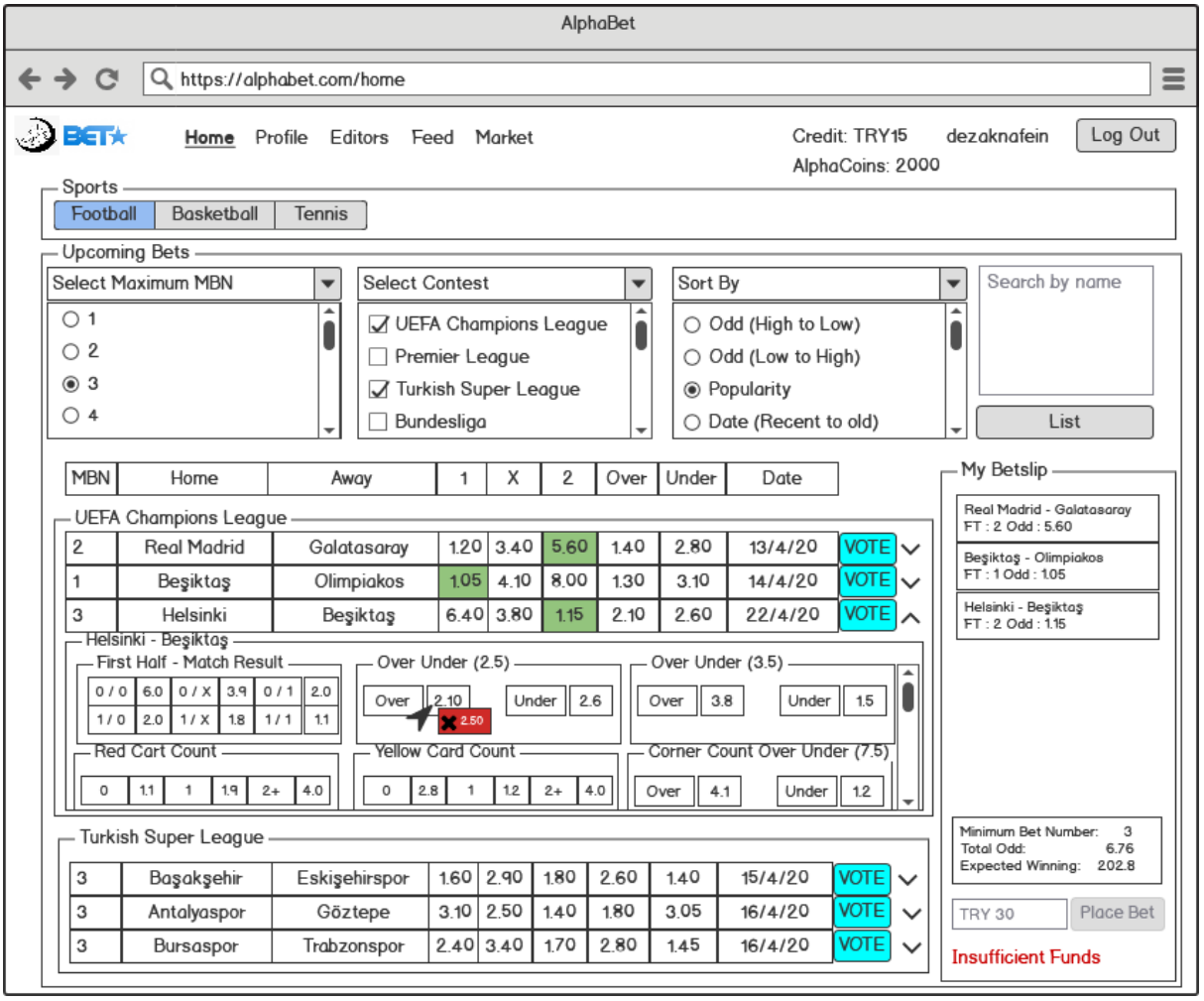
Candidate Keys: {(user_id)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bans(
user_id INT,
admin_id INT,
PRIMARY KEY(user_id, admin_id),
FOREIGN KEY(user_id) REFERENCES user(user_id) ON
DELETE CASCADE,
FOREIGN KEY(admin_id) REFERENCES admin(admin_id)
ON DELETE CASCADE
);
```

3. User Interface and SQL Queries

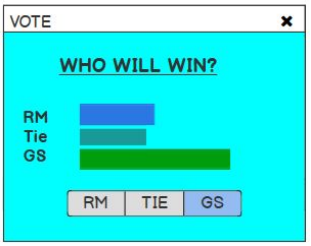
3.1.Home Page



Inputs: @username, @password, @search_text, @sport_name, @contest_choice, @bet_type, @sort_type, @mbn, @bet_id, @match_id, @played_amount, @vote_side

Process: Once a user opens AlphaBet, the user will be welcomed with the home screen. A user can login to the system by entering username and password and then pressing the “Login” button. If the user is not already signed up, the user will be forwarded to the Sign Up screen after pressing the “Sign Up” button. Here, a user can filter the bets in order to narrow down the search. First, the user must select a sport, and then the user can choose a bet related to that sport, a contest related to that sport and can sort the bets by date, popularity (number of people who invested on that bet), and odds. If the user does not choose to select a filter, say contest filter, it will be assumed that everything will be included in the result from that particular filter. Users can also search for a team name in order to view a bet including that team name. After the user filters the bets, “List” button must be pressed to apply these filters. Users can click on a bet in order to display all the odds related to that bet. If an odd has been changed by the

admin, the user will be able to see that as a small tooltip. Users can also click on the vote icon in order to vote for the team that they think will win. Vote results will be displayed afterwards, as seen in the figure on the right. On the bottom right corner, users can see their current betslip with bets and necessary information like mbn, total odd and expected winnings. Users can then enter an amount in order to place this betslip. If, however, the minimum bet number was not satisfied or the user account does not contain the amount that was entered, it will print an error message. Up in the toolbar, users can navigate to “Profile”, “Editors” and “Feed” pages, which will be explained in detail.



SQL Statements:

Filtering bets and matches using keywords and selections:

WITH sport_filter AS (SELECT match_id FROM match WHERE sport_name = “@sport_name”),

bet_filter AS (SELECT match_id FROM bet WHERE active = “true” AND mbn <= @mbn),

contest_filter AS (SELECT match_id FROM match NATURAL JOIN contest WHERE contest_name IN @contest_choice),

individual_player_filter AS (SELECT match_id FROM competitor_match NATURAL JOIN individual_player WHERE forename LIKE “%@search_text%” OR surname LIKE “%@search_text%”),

team_filter AS (SELECT match_id FROM competitor_match NATURAL JOIN team WHERE team_name LIKE “%@search_text%”),

final_filter AS (SELECT match_id FROM sport_filter INTERSECT bet_filter INTERSECT contest_filter INTERSECT individual_player_filter INTERSECT team_filter)

SELECT * FROM final_filter

Show all possible bets of a particular sport using the filter:

WITH match_data AS (SELECT * FROM final_filter NATURAL JOIN match),

all_competitors AS (SELECT competitor_name, competitor_id FROM (SELECT player_id AS competitor_id, CONCAT(forename, ' ', surname) AS competitor_ name FROM individual_player) AS temp UNION (SELECT team_id AS competitor_id, team_name AS competitor_name FROM team)),

current_competitors AS (SELECT competitor_id, side, match_id FROM competitor_match NATURAL JOIN final_filter),

```
all_sides AS (SELECT competitor_name, side, match_id FROM all_competitors
NATURAL JOIN current_competitors),

bet_data AS (SELECT * FROM bet NATURAL JOIN final_filter),

old_odds AS (SELECT match_id, bet_type, MAX(change_date) AS change_date
FROM (SELECT * FROM bet NATURAL JOIN final_filter WHERE active = 'false') AS
inactive_bets GROUP BY match_id, bet_type)

SELECT * FROM match_data NATURAL JOIN bet_data NATURAL JOIN all_sides
NATURAL JOIN old_odds
```

Find popular bets:

```
WITH active_bet AS ( SELECT bet_id, match_id FROM bet WHERE active = ‘true’),

all_placed_bets AS ( SELECT * FROM included_bet NATURAL JOIN bet_slip
NATURAL JOIN active_bet WHERE placed = ‘true’)

SELECT Count(bet_slip_id) as  FROM all_placed_bets GROUP BY  bet_id, match_id
```

Creating the initial bet slip:

```
INSERT INTO bet_slip (creator_id, placed, played_amount) VALUES ( SELECT
person_id FROM person WHERE username = @username, FALSE, 0)
```

Selecting bet and adding to betslip:

```
INSERT INTO included_bet (bet_slip_id, bet_id, match_id) VALUES (SELECT
bet_slip_id FROM bet_slip WHERE placed = FALSE AND creator_id  = (SELECT
person_id FROM person WHERE username = @username) ), @bet_id, @match_id)
```

Check MBN criteria of bet slip:

```
WITH user_bet_slip AS (SELECT bet_slip_id FROM bet_slip WHERE creator_id =
(SELECT person_id FROM person WHERE username = @username) AND placed =
FALSE ),
user_current_bets AS (SELECT * FROM user_bet_slip NATURAL JOIN included_bet
NATURAL JOIN bet ),
cur_no_bet AS (SELECT Count(bet_slip_id) AS bet_count FROM
user_current_bets),
max_mbn_no AS (SELECT Max(mbn) AS max_mbn FROM user_current_bets)
SELECT CASE
WHEN cur_no_bet.bet_count < max_mbn_no.max_mbn THEN 'MBN not satisfied!''
WHEN cur_no_bet.bet_count >= max_mbn_no.max_mbn THEN 'MBN is satisfied!''
        END AS response
FROM cur_no_bet, max_mbn_no
```

Check if user account balance is enough:

```
SELECT CASE
        WHEN user.account_balance < 3 THEN 'Insufficent credits'
        WHEN user.account_balance > 3 THEN 'User has enough credits.'
        END AS response
FROM user WHERE user_id = (SELECT person_id FROM person WHERE
username = @username)
```

Placing the desired amount of money on the bet slip:

```
UPDATE bet_slip
SET played_amount = @played_amount
WHERE creator_id = (SELECT person_id FROM person WHERE username =
@username)
```

Placing the betslip (assuming MBN and account balance queries above are checked beforehand):

```
UPDATE bet_slip
SET placed = TRUE
WHERE creator_id = (SELECT person_id FROM person WHERE username =
@username)

UPDATE user
SET alpha_coins = @played_amount * 3
WHERE user_id = (SELECT person_id FROM person WHERE username =
@username)
```

User votes for a side or tie:

```
INSERT INTO votes (match_id, user_id, side) VALUES (@match_id, SELECT
person_id FROM person WHERE username = @username, @vote_side)
```

Calculate home side votes:

```
SELECT COUNT(side) AS home_vote_count FROM votes GROUP BY match_id
HAVING side = 'HOME'
```

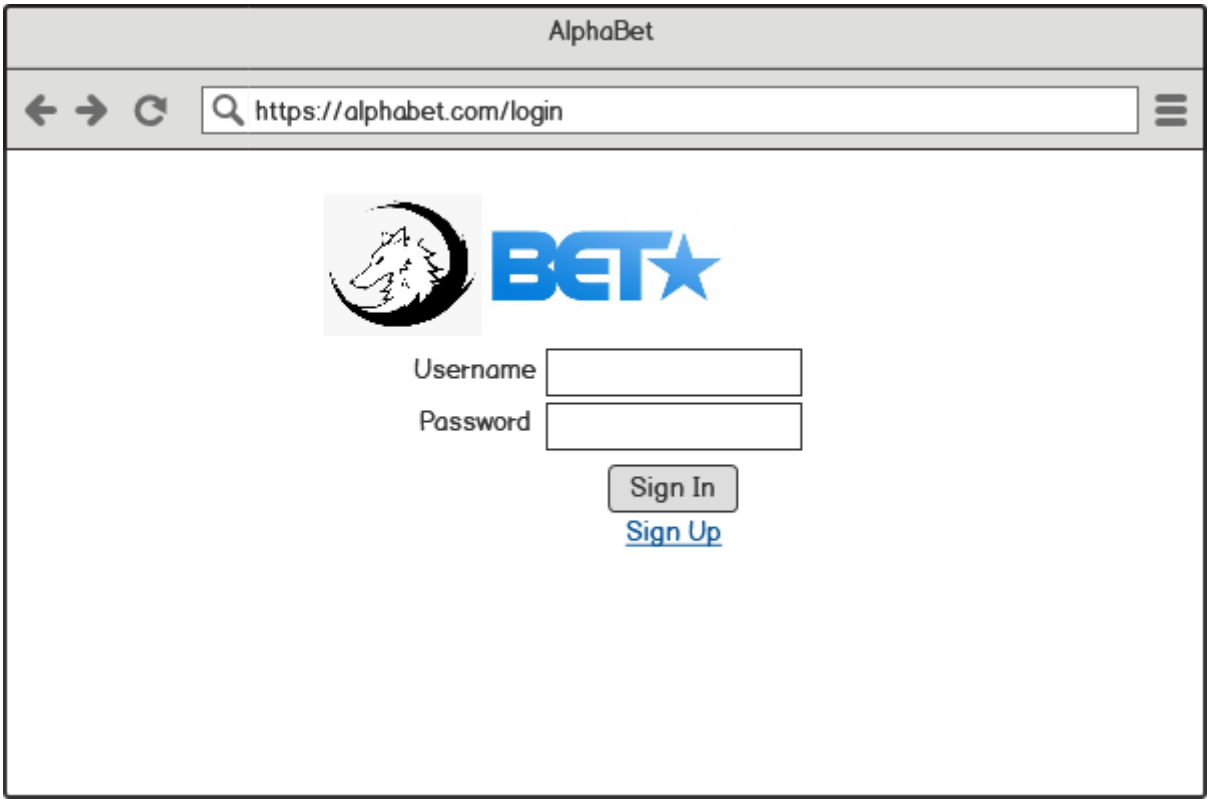
Calculating away side votes:

```
SELECT COUNT(side) AS away_vote_count FROM votes GROUP BY match_id
HAVING side = 'AWAY'
```

Calculating tie votes:

```
SELECT COUNT(side) AS tie_vote_count FROM votes GROUP BY match_id
HAVING side = 'TIE'
```

3.2.Login Page



Inputs: @username, @password

Process: The user enters their email address and password to login to AlphaBet Social Betting Platform. If the user is not already signed up to the system, the user can signup by pressing the “Sign Up” button. After the user enters credentials, the system checks if the user is banned.

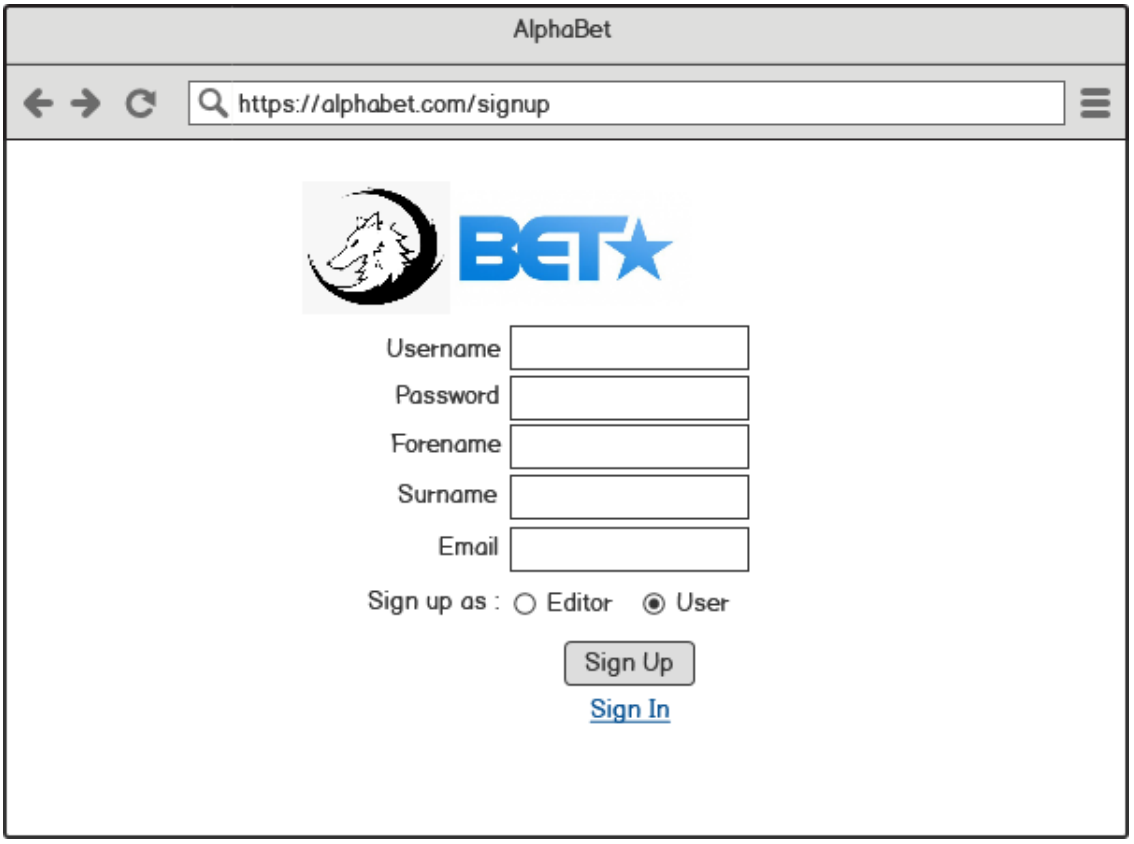
SQL Statements:

Login:

WITH this_user AS (SELECT user_id FROM person WHERE username = @username)

SELECT username, user_id FROM person WHERE username = @username AND password = @password AND NOT EXISTS (SELECT user_id FROM bans WHERE user_id = this_user) AND NOT EXISTS (SELECT editor_id FROM editor_request WHERE editor_id = this_user AND state = ‘PENDING’)

3.3.Sign Up Page



Inputs: @username, @password, @forename, @surname, @email

Process: User will enter a unique username along with password, forename, surname and email.A person can choose either editor or user, but initially they will just be added to the database as a person. If a person clicks on the editor button, the admin will receive a notification and the admin will need to verify this information before one can sign up as an editor.

SQL Statements:

Add user:

INSERT INTO person (username, password, forename, surname, e-mail) VALUES (@username, @password, @forename, @surname, @email);

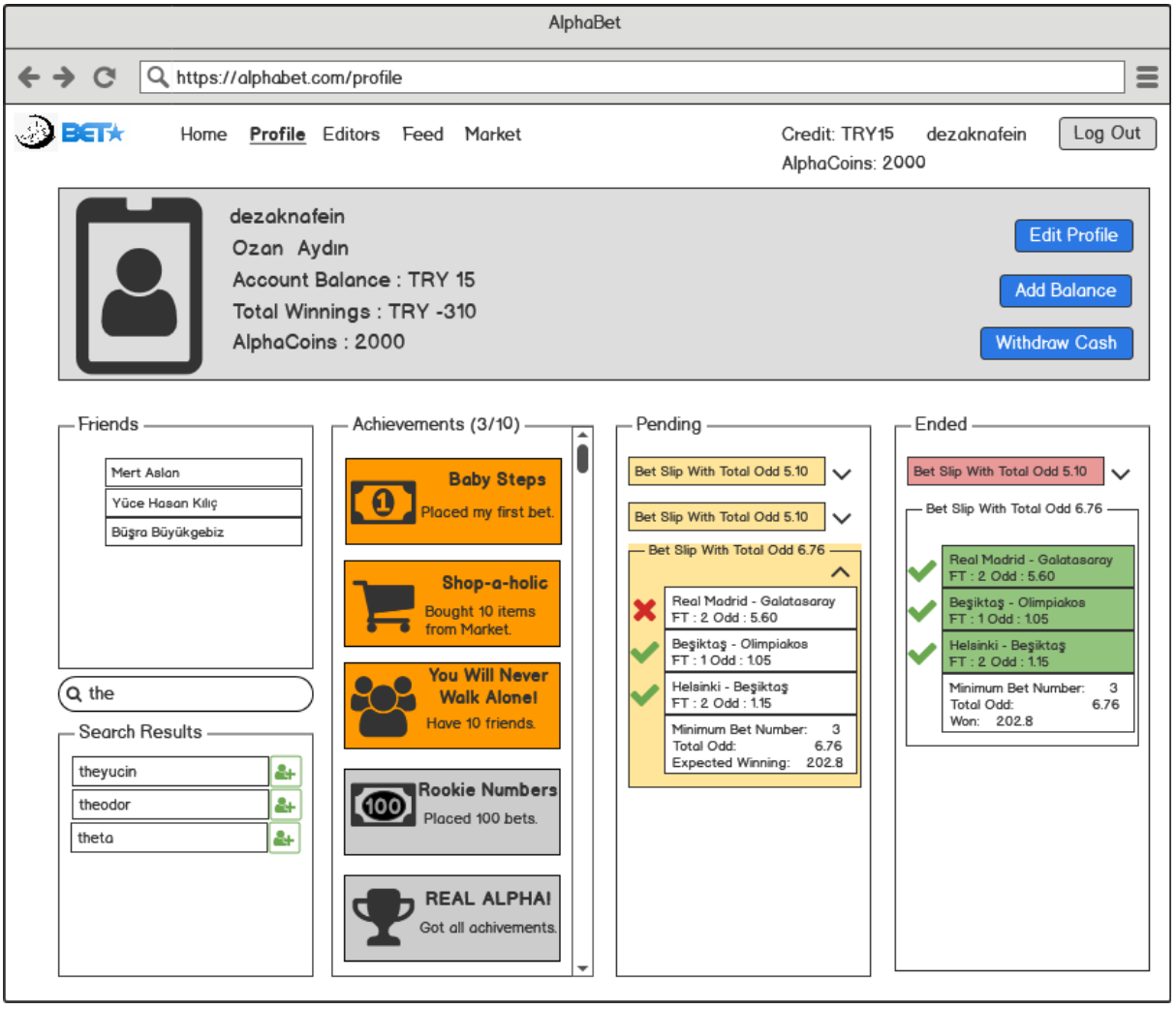
INSERT INTO user (account_balance, total_winnings) VALUES (0, 0);

Add editor:

INSERT INTO person (username, password, forename, surname, e-mail) VALUES (@username, @password, @forename, @surname, @email);

INSERT INTO editor_request (editor_id, admin_id, state) VALUES (SELECT person_id FROM person WHERE username = @username, NULL, ‘PENDING’)

3.4.Profile Page



Inputs: @user_id, @new_username, @new_password, @friend_id, @balance_change, @friend_search_text

Process: In the profile page, users can view various information. Users can see their friends and search for their friends by name. Users can collect achievements and see their collected achievements on their profile page. Pending bet slips and completed bets will be visible as well. Users can also edit their profile (change username, password and full name), add their balances and withdraw cash from their balances.

SQL Statements:

Display Friends:

SELECT username FROM ((SELECT user_friend.friend_id AS “user_id” FROM user_friend WHERE user_friend.user_id = @user_id) NATURAL JOIN user);

Display Pending Bet Slips:

```
WITH user_bet_slips AS (SELECT bet_slip_id FROM bet_slip WHERE
creator_id = @user_id AND placed = TRUE),

pending_slip AS (SELECT bet_slip_id FROM (user_bet_slips NATURAL JOIN
included_bet) AS keys NATURAL JOIN bet) WHERE result = 'PENDING'),

all_bet_data AS (SELECT * FROM (pending_slip NATURAL JOIN included_bet)
AS bet_keys NATURAL JOIN bet),

match_data AS (SELECT * FROM all_bet_data NATURAL JOIN
competitor_match),

all_competitors AS (SELECT competitor_name, competitor_id FROM (SELECT
player_id AS competitor_id, CONCAT(forename, ' ', surname) AS competitor_
name FROM individual_player) AS temp UNION (SELECT team_id AS
competitor_id, team_name AS competitor_name  FROM team)),

SELECT * FROM match_data NATURAL JOIN all_competitors
```

Display Ended Bet Slips:

```
WITH user_bet_slips AS (SELECT bet_slip_id FROM bet_slip WHERE
creator_id = @user_id),

ended_slip AS (SELECT bet_slip_id FROM (user_bet_slips NATURAL JOIN
included_bet) AS keys NATURAL JOIN bet) WHERE result = 'WON' OR result =
'LOST'),

all_bet_data AS (SELECT * FROM (ended_slip NATURAL JOIN included_bet)
AS bet_keys NATURAL JOIN bet),

match_data AS (SELECT * FROM all_bet_data NATURAL JOIN
competitor_match),

all_competitors AS (SELECT competitor_name, competitor_id FROM (SELECT
player_id AS competitor_id, CONCAT(forename, ' ', surname) AS competitor_
name FROM individual_player) AS temp UNION (SELECT team_id AS
competitor_id, team_name AS competitor_name  FROM team)),

SELECT * FROM match_data NATURAL JOIN all_competitors
```

Display Gained Achievements:

```
WITH achieved_id AS (SELECT achievement_id FROM gained_achievement
NATURAL JOIN user WHERE user_id = @user_id),

SELECT * FROM achieved_id NATURAL JOIN achievement
```

Display Total Achievement Count:
SELECT COUNT(achievement_id) AS total_count FROM achievement GROUP BY achievement_id

Display Gained Achievement Count:
WITH achieved_id AS (SELECT achievement_id FROM gained_achievement NATURAL JOIN user WHERE user_id = @user_id),

SELECT COUNT(achievement_id) AS gained_count FROM achieved_id GROUP BY achievement_id

Display User Info:
WITH user_person AS ((SELECT user_id AS person_id FROM user WHERE user_id = @user_id) NATURAL JOIN person),

SELECT username, forename, surname, account_balance, total_winnings, alpha_coins FROM user_person

Search friends:
SELECT username FROM user AS u INNER JOIN person AS p ON u.user_id = p.person_id WHERE username LIKE '%@friend_search_text%' AND u.user_id <> @user_id

Add friend:
INSERT INTO user_friend (user_id, friend_id) VALUES (@user_id, @friend_id)

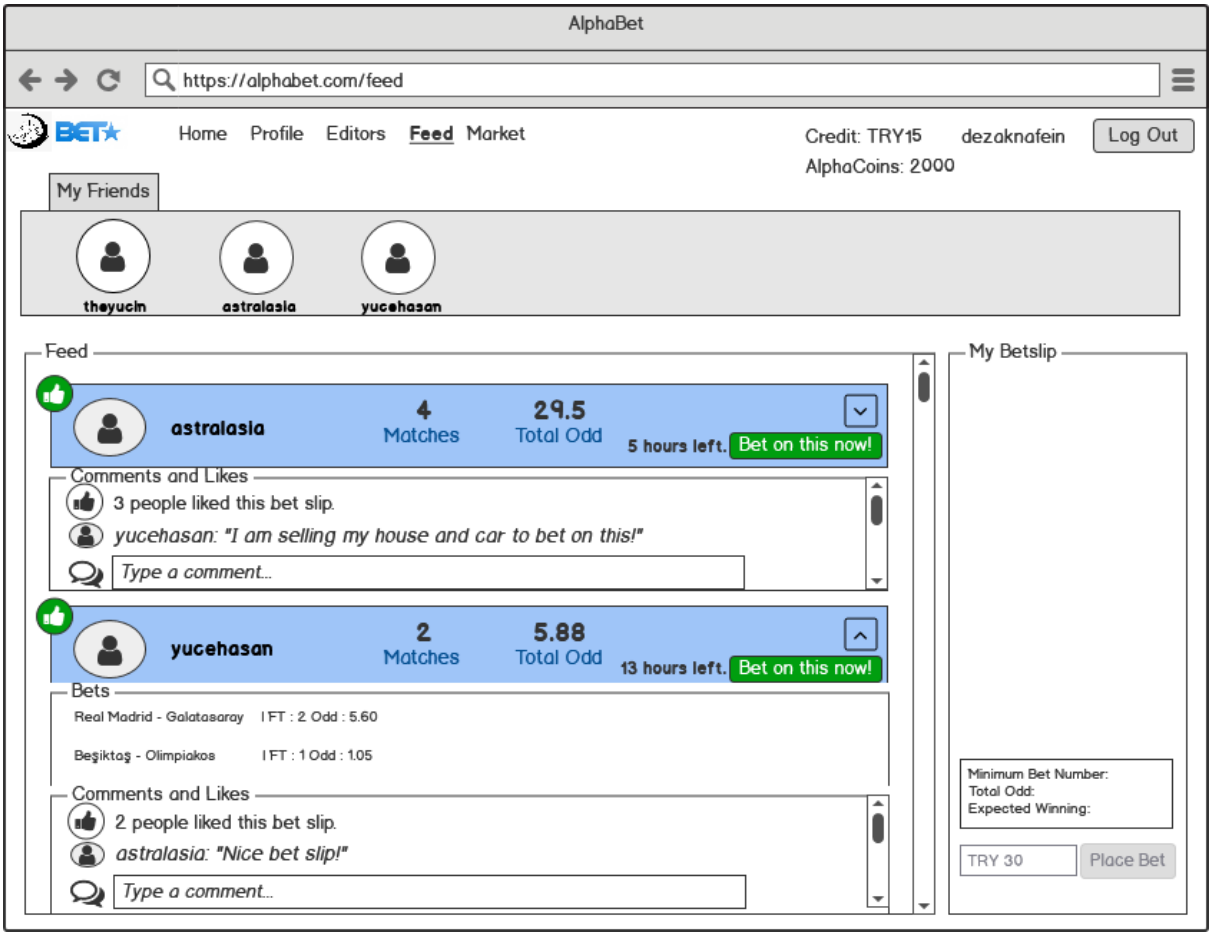
Add balance / Withdraw Cash:
UPDATE user
SET account_balance = (account_balance + @balance_change)
WHERE user_id = @user_id

Change username / password:
WITH taken_username AS(SELECT username FROM person)

UPDATE person
SET username = @new_username
WHERE @new_username NOT IN taken_username AND person.person_id = @user_id

UPDATE person
SET password = @new_password
WHERE @new_password <> (SELECT password FROM person WHERE person.person_id = @user_id) AND person.person_id = @user_id

3.5.Feed Page



Inputs: @comment_text, @comment_id, @user_id, @focused_bet_slip_id, @date

Process: Feed page allows users to see their friends' shared bets slips and comments. Users can see a shared bet slip, like it, comment on it, and click the button “Bet on this now” to add that bet slip into their own bet slip.

SQL Statements:

Display bet slips shared by friends:

WITH friend_id_set AS (SELECT friend_id AS user_id FROM user_friend WHERE user_id = @user_id),

friend_data AS(SELECT username, person_id AS sharer_id FROM friend_id_set NATURAL JOIN person),

friend_slip_id AS (SELECT * FROM (shared_bet_slip NATURAL JOIN (SELECT user_id AS sharer_id FROM friend_id_set) AS sharing_friend)),

```
    friend_slip_bet AS (SELECT * FROM (included_bet NATURAL JOIN
friend_slip_id)),

    friend_slip_bet_data AS (SELECT * FROM friend_slip_bet NATURAL JOIN
bet),

match_data AS (SELECT * FROM friend_slip_bet_data NATURAL JOIN
competitor_match),

all_competitors AS (SELECT competitor_name, competitor_id FROM (SELECT
player_id AS competitor_id, CONCAT(forename, ' ', surname) AS competitor_
name FROM individual_player) AS temp UNION (SELECT team_id AS
competitor_id, team_name AS competitor_name  FROM team)),

SELECT * FROM match_data NATURAL JOIN all_competitors NATURAL JOIN
(SELECT sharer_id AS user_id, username FROM friend_data)
```

User likes a bet slip:

```
INSERT INTO bet_slip_like (user_id, bet_slip_id) VALUES (@user_id,
@focused_bet_slip_id)
```

User comments on a betslip:

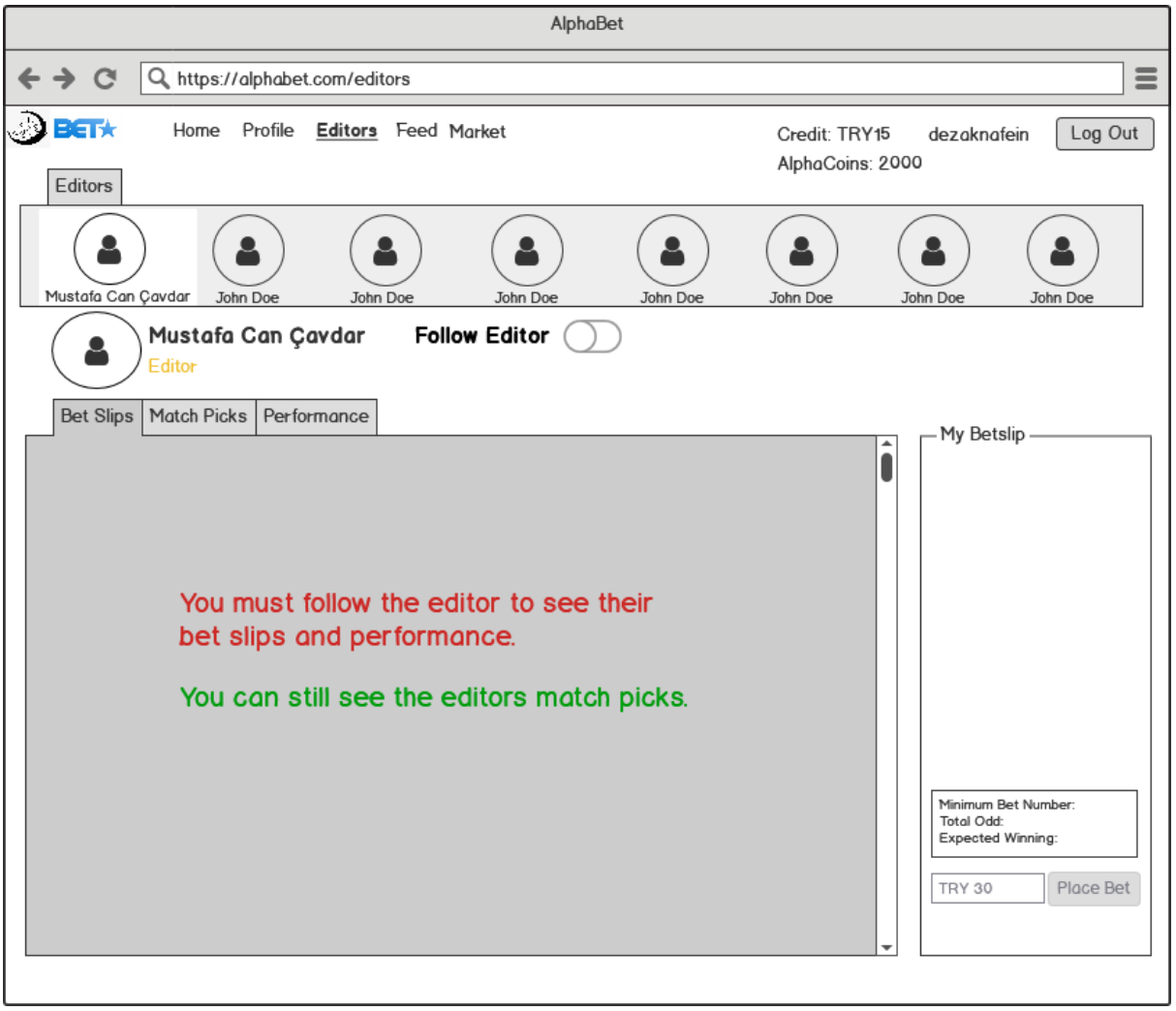
```
INSERT INTO comment (comment, person_id, comment_date) VALUES
(@comment_text, @user_id, @date)

DECLARE @inserted_comment_id INT
SET @inserted_comment_id = SCOPE_IDENTITY()

INSERT INTO bet_slip_comment(comment_id, bet_slip_id) VALUES
(@inserted_comment_id, @focused_bet_slip_id)
```

User deletes a comment:

```
DELETE FROM bet_slip_comment WHERE (comment_id = @comment_id AND
user_id = @user_id)
DELETE FROM comment WHERE comment_id = @comment_id
```

3.6.Editor Page

3.6.1. Unfollowed Editor Page



Inputs: @editor_id, @user_id

Process: The “Editors” page opens with the first editor selected. A user can choose an editor from all editors available and follow them on their page using the “Follow Editor” button. The user must follow the editor to see their bet slips and performance but single bet suggestions of the editor can still be seen by everybody.

SQL Statements:

Display editor information:

SELECT * FROM editor WHERE editor_id = @editor_id

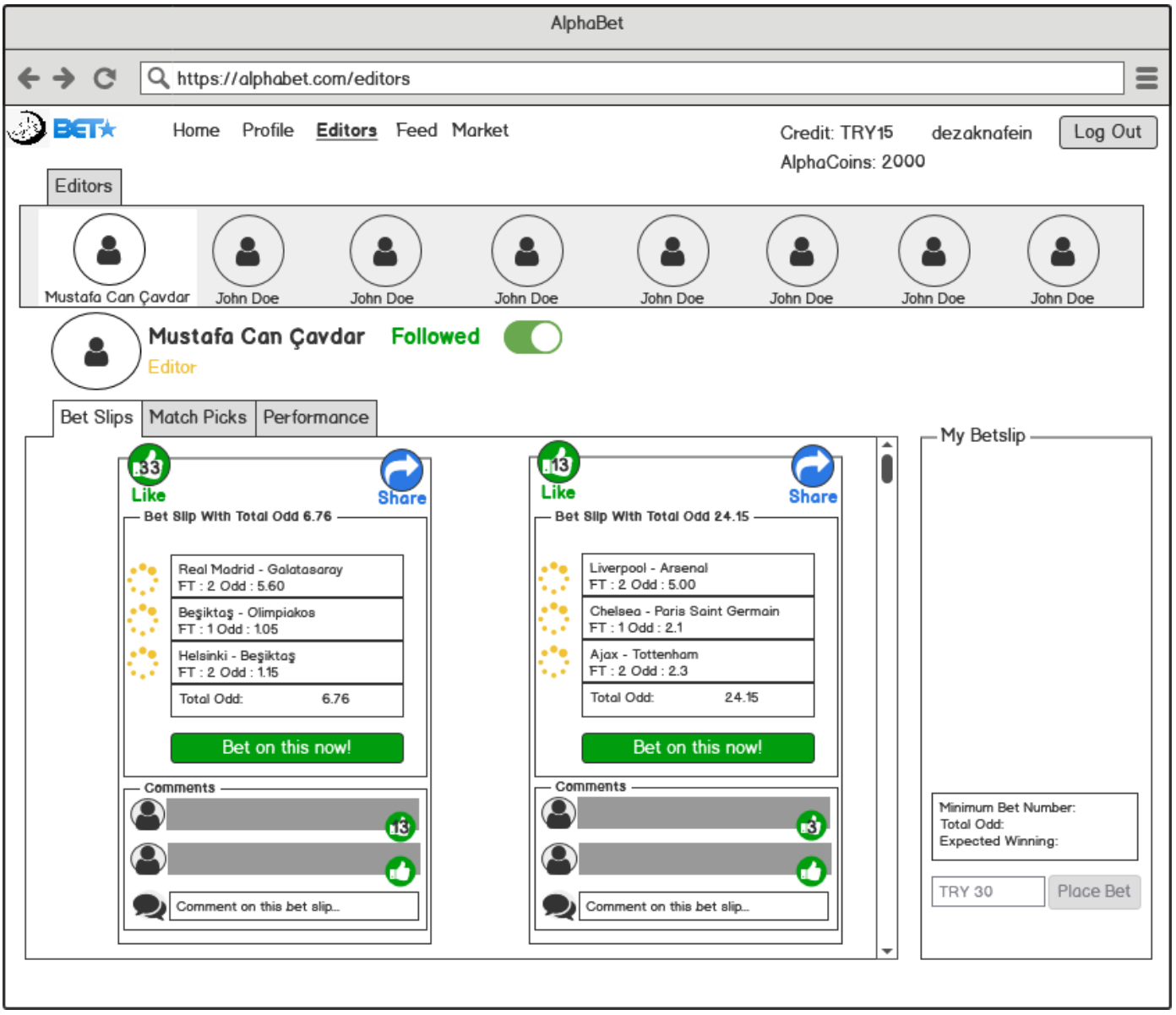
Display editors:

SELECT forename, surname FROM ((SELECT editor.editor_id AS “person_id” FROM editor) NATURAL JOIN person);

Follow editor:
INSERT INTO user_follows (editor_id, user_id) VALUES (@editor_id, @user_id);

Unfollow editor:
DELETE FROM user_follows WHERE user_id = @user_id AND editor_id = @editor_id;

3.6.2. Editors Bet Slips Page



Inputs: @editor_id, @user_id, @editor_bet_slip_id, @comment_id, @comment_text, @date

Process: This is the bet slip of the editor that a user follows. Here, users can see the bet slips that the editors have shared. A user can like a particular bet slip of an editor,

share that bet slip with their friends and comment on a bet that the editor shared. It is also possible to like a comment. Users can copy the editor's bet slip to their own bet slip by clicking on the "Bet on this now" button. It should be noted that users can add other bets on top of the copied bet slip of the editor. The input named "comment_id" will be received when the user clicks on a bet in order to copy the content into their bet slip.

SQL Statements:

Display bet slips that the editor shared:

```
SELECT * FROM ((SELECT * FROM shared_bet_slip WHERE sharer_id =
@editor_id) NATURAL JOIN included_bets NATURAL JOIN bet) WHERE
EXISTS (SELECT editor_id FROM user_follows WHERE user_id = @user_id
AND editor_id = @editor_id);
```

User likes a particular bet slip of the editor:

```
INSERT INTO bet_slip_like (user_id, bet_slip_id) VALUES (@user_id,
@editor_bet_slip_id)
```

User shares a particular bet slip of the editor:

```
INSERT INTO shared_bet_slip (bet_slip_id, sharer_id) VALUES
(@editor_bet_slip_id, @user_id)
```

User comments a particular bet slip of the editor:

```
INSERT INTO comment (comment, person_id, comment_date) VALUES
(@comment_text, @user_id, @date)

DECLARE @inserted_comment_id INT
SET @inserted_comment_id = SCOPE_IDENTITY()

INSERT INTO bet_slip_comment(comment_id, bet_slip_id) VALUES
(@inserted_comment_id, @bet_slip_id)
```

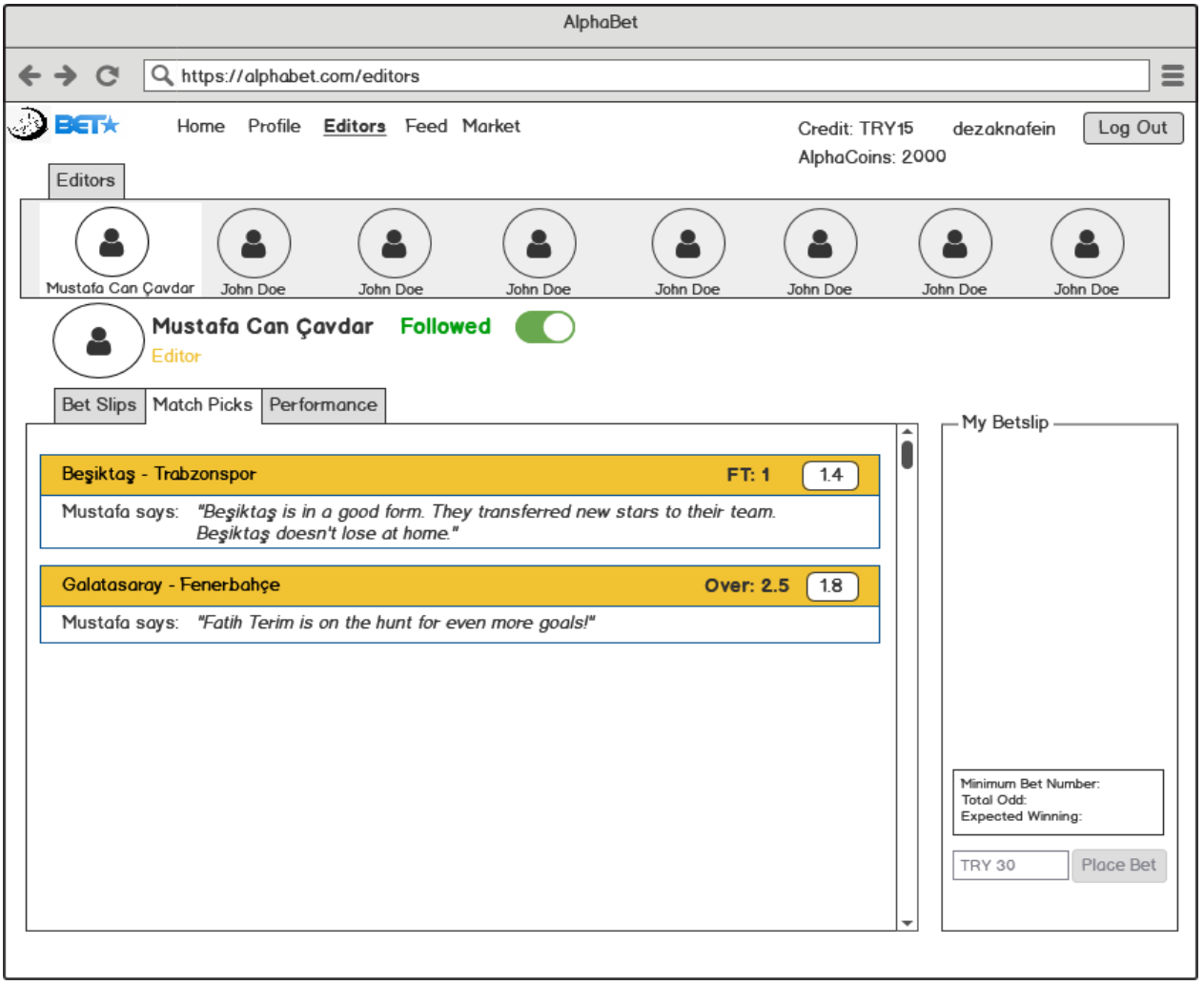
User deletes comment:

```
DELETE FROM bet_slip_comment WHERE (comment_id = @comment_id
AND user_id = @user_id)
DELETE FROM comment WHERE comment_id = @comment_id
```

Betting on the editor's bet slip:

```
WITH all_bet_of_bet_slip AS (SELECT * FROM (SELECT match_id, bet_id
FROM included_bet WHERE bet_slip_id = @editor_bet_slip_id) AS bets JOIN
(SELECT bet_slip_id FROM bet_slip WHERE creator_id = @user_id AND
placed = FALSE) AS my_bet_slip_id ),

INSERT INTO included_bet (match_id, bet_id, bet_slip_id)
all_bet_of_bet_slip
```

3.6.3. Editors Match Picks Page



Inputs: @editor_id, @bet_id, @match_id

Process: Users can access this page by clicking on “Match Picks” in the editor page, and users who are not following the current editor are also able to view this page. Here, single match picks of the editor are visible, along with a comment from the editor, detailing the reasons behind their choice. Users can click on one of these bets in order to add that bet to their bet slip.

SQL Statements:

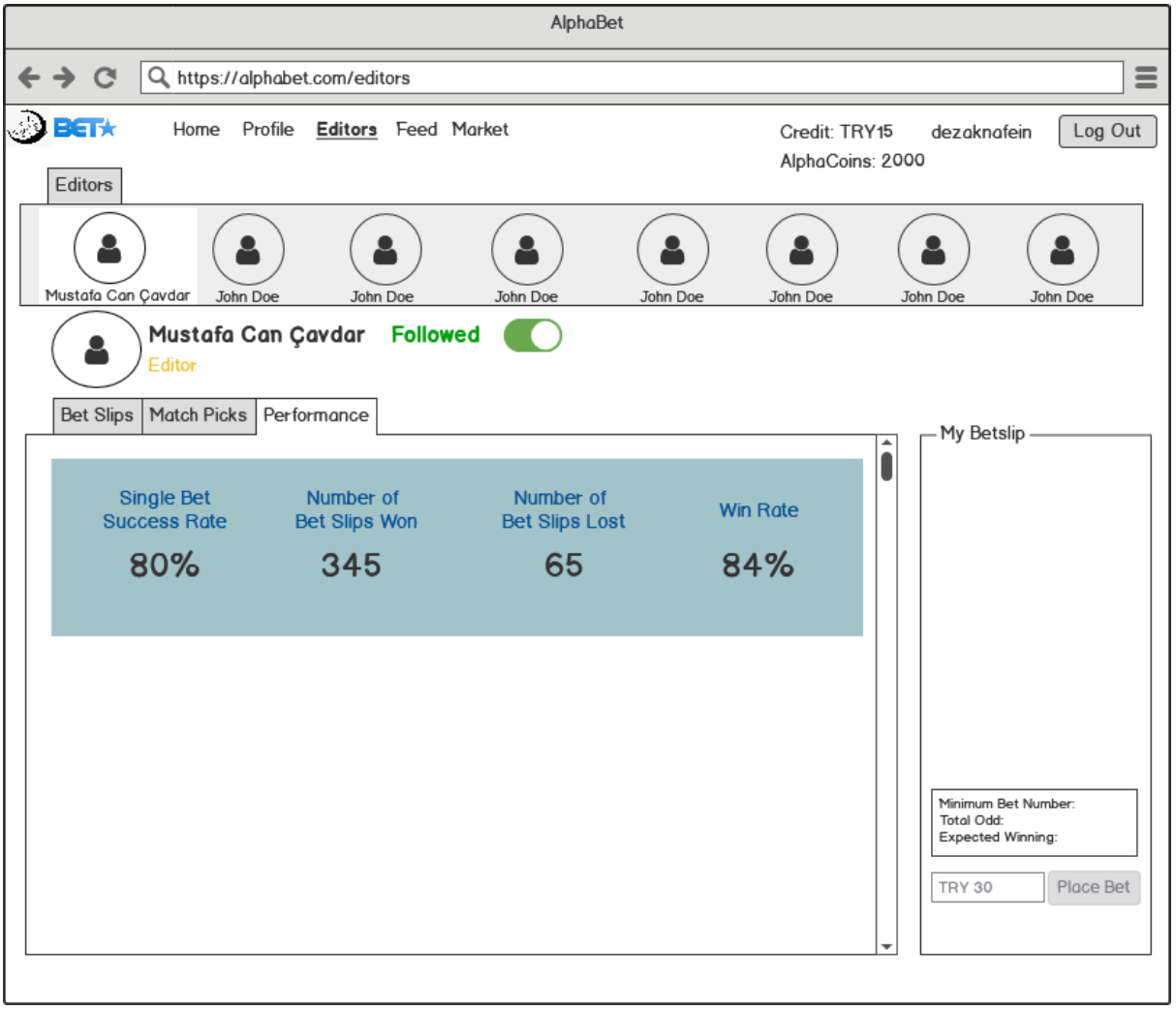
Display bets:

SELECT * FROM ((SELECT * FROM suggested_bet WHERE editor_id = @editor_id) NATURAL JOIN bet)

Add single bet to the user’s bet slip:

INSERT INTO included_bet (bet_slip_id, bet_id, match_id) VALUES (SELECT bet_slip_id FROM bet_slip WHERE placed = FALSE AND creator_id = @editor_id), @bet_id, @match_id)

3.6.4. Editors Performance Page



Inputs: @editor_id, @user_id

Process: Users can access this page by clicking on the “Performance” button, however this page is only visible to the users that follow this editor. Here, the single bet success rate is shown, calculated from the success rate of their single match reviews. Also, the total number of successful and unsuccessful bet slips that the editor shared is visible. Win rate is just winning bet slips over total bet slips multiplied by 100.

SQL Statements:

Single bet success rate:

```
WITH editor_bets AS (SELECT bet_id, match_id FROM suggested_bet
WHERE editor_id = @editor_id)

SELECT COUNT(bet_id) AS won_count FROM (editor_bets NATURAL JOIN
bet) AS joined_editor_bets WHERE joined_editor_bets.result = ‘won'
```

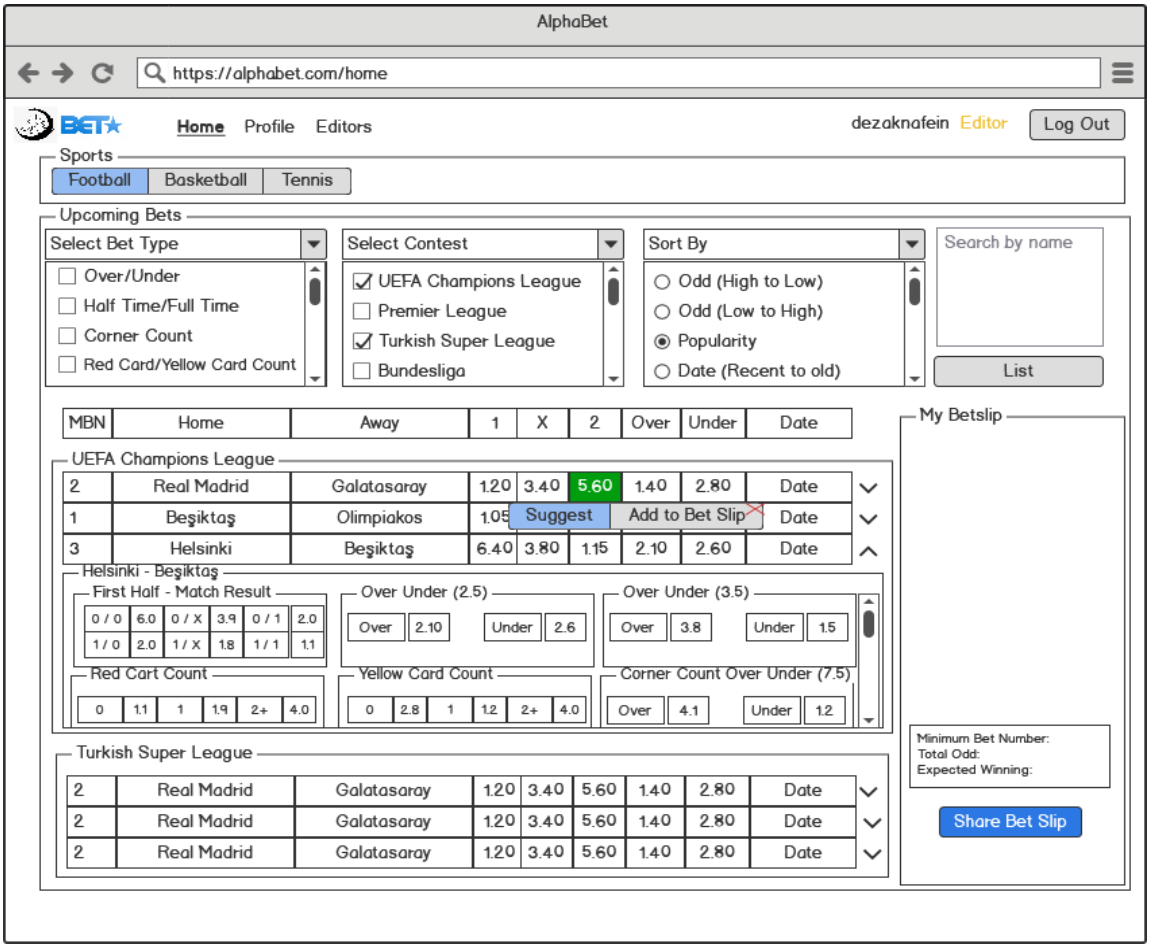
Number of bet slips won:

```
WITH editor_slips AS (SELECT bet_slip_id FROM bet_slip WHERE
creator_id = @editor_id AND @editor_id IN (SELECT editor_id FROM
user_follows WHERE user_id = @user_id)),

editor_bet_id AS (SELECT * FROM included_bet NATURAL JOIN
editor_slips)

SELECT COUNT(bet_slip_id) AS won_bet_slip_count FROM editor_slips
WHERE NOT EXISTS (SELECT bet_id, match_id FROM (editor_bet_id
NATURAL JOIN bet) AS editor_bets WHERE editor_slips.bet_slip_id =
editor_bets.bet_slip_id AND (editor_bets.result = 'lost' OR editor_bets.result =
'pending'))
```

Number of bet slips lost:

```
WITH editor_slips AS (SELECT bet_slip_id FROM bet_slip WHERE
creator_id = @editor_id AND @editor_id IN (SELECT editor_id FROM
user_follows WHERE user_id = @user_id)),

editor_bet_id AS (SELECT * FROM included_bet NATURAL JOIN
editor_slips)

SELECT COUNT (bet_slip_id ) FROM editor_slips WHERE EXISTS
(SELECT bet_id, match_id FROM (editor_bet_id NATURAL JOIN bet) AS
editor_bets WHERE editor_slips.bet_slip_id = editor_bets.bet_slip_id AND
(editor_bets.result = 'lost')
```

3.6.5. Editor Home Page



Inputs: @editor_id, @bet_slip_id, @bet_id, @match_id, @comment_text

Process: An editor sees Home Page as in the figure. By pressing on an odd of a match, a popup appears including two buttons, “Suggest” and “Add to Bet Slip”. If the editor presses on the “Add to Bet Slip” button, the bet is added to the editors bet slip. If the editor presses on the “Suggest” button, an input text field pops up under the suggest button where the editor is able to type their comment on the suggested bet and then the bet is shared on the editors “Match Picks” section. By pressing the “Share Bet Slip” button, the editor can share the bet slip s/he created.

SQL Statements:

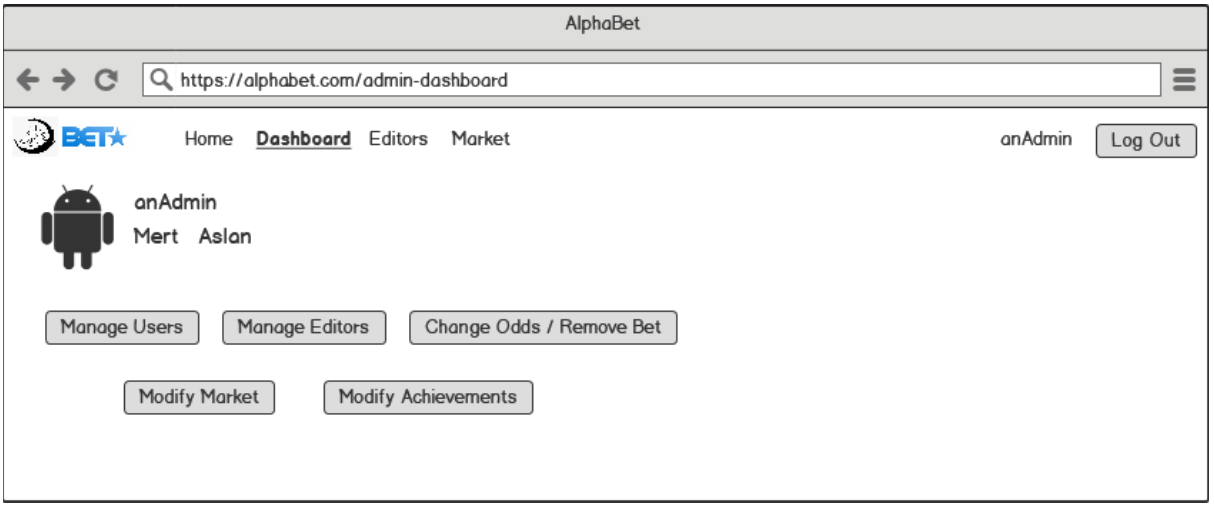
Editor shares a bet slip:

INSERT INTO shared_bet_slip (bet_slip_id, sharer_id) VALUES (@bet_slip_id, @editor_id)

Editor suggests a bet slip:

INSERT INTO suggested_bet(editor_id, bet_id, match_id, comment) VALUES (@editor_id, @bet_id, @match_id, @comment_text)

3.7. Admin Dashboard Page



Inputs: @admin_id

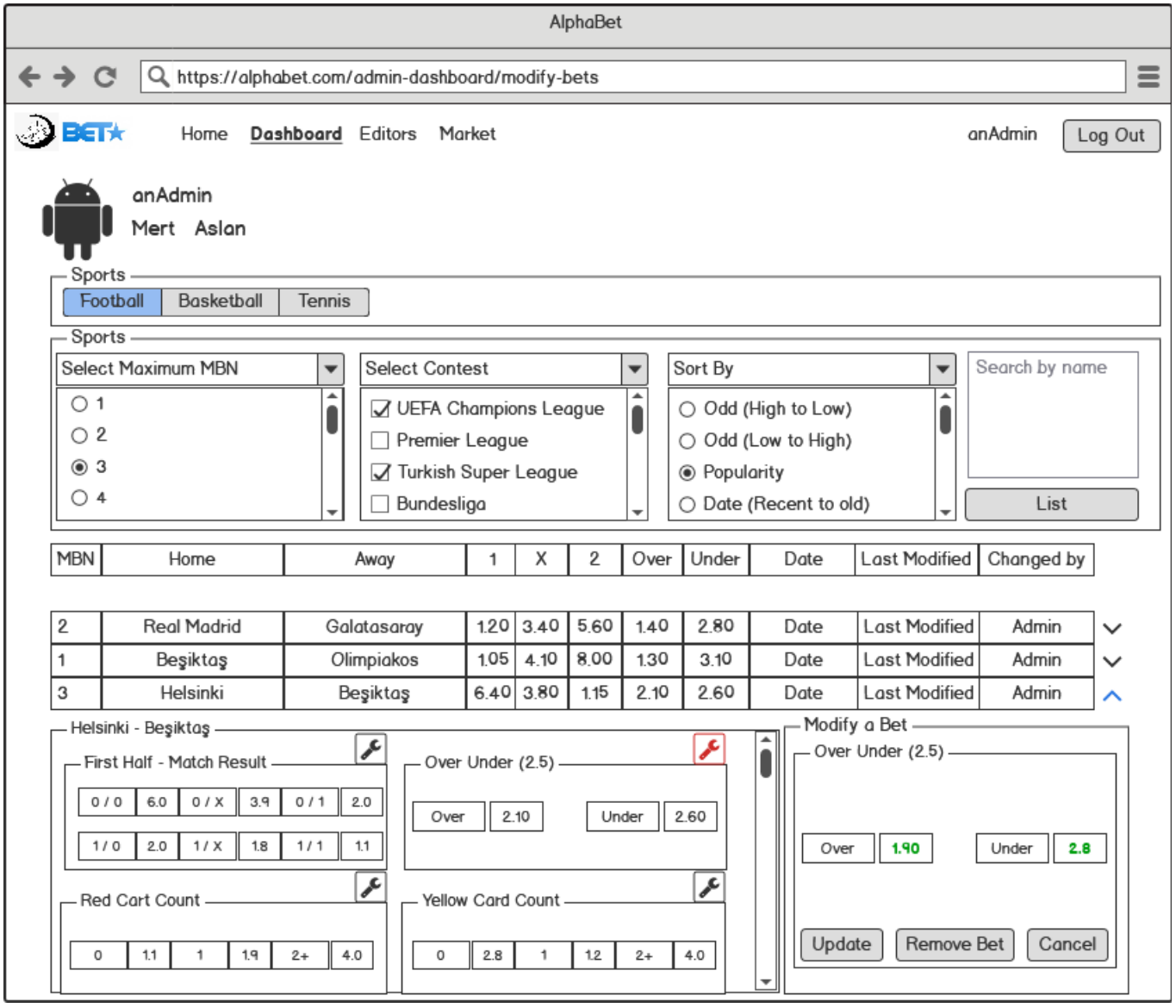
Process: The admin can manage users, editors and bets on the dashboard screen. After pressing on the “Change Odds/Remove Bet” button the admin is forwarded to a page where s/he can see, modify and remove all possible bets. After pressing the “Manage Editors” button the admin is forwarded to a page where s/he can evaluate editor applications. After pressing the “Manage Users” button, the admin is forwarded to another page where s/he can block and unblock users on the system. The admin can modify the achievements by adding/removing achievements using the “Modify Achievements” button. And finally with the “Modify Market” button, admin can modify the market page by adding/removing items.

SQL Statements:

Display Admin Information:

SELECT * FROM admin a INNER JOIN person p ON a.admin_id = p.person_id

3.7.1. Admin Manage Bets Page



Inputs: @new_odd_value, @bet_id, @match_id, @bet_type, @mbn

Process: Admin can view this page by clicking the “Change Odds/ Remove Bet” button in the admin dashboard. In this page, the admin can filter or search for the bet that he wants to make changes on (the same filtering algorithm that is being used in the home page applies), and then by clicking on the Edit icon on the top right corner of each odd, he can enter new odds and update the bet, or he can remove the bet completely.

SQL Statements:

List all possible bets with possible features:

WITH sport_filter AS (SELECT match_id FROM match WHERE sport_name = “@sport_name”),

```
bet_filter AS (SELECT match_id FROM bet WHERE active = “true” AND mbn
<= @mbn),

contest_filter AS (SELECT match_id FROM match NATURAL JOIN contest
WHERE contest_name IN @contest_choice),

individual_player_filter AS (SELECT match_id FROM competitor_match
NATURAL JOIN individual_player WHERE forename LIKE
“%@search_text%” OR surname LIKE “%@search_text%”),

team_filter AS (SELECT match_id FROM competitor_match NATURAL JOIN
team WHERE team_name LIKE “%@search_text%”),

final_filter AS (SELECT match_id  FROM sport_filter INTERSECT bet_filter
INTERSECT contest_filter INTERSECT individual_player_filter INTERSECT
team_filter),

match_data AS (SELECT * FROM final_filter NATURAL JOIN match),

all_competitors AS (SELECT team_name AS competitor_name,
competitor_id FROM (SELECT competitor_id, CONCAT(forename, ' ',
surname) AS team_name FROM individual_player) AS temp UNION
(SELECT * FROM team)),

current_competitors AS (SELECT competitor_id, side, match_id FROM
competitor_match NATURAL JOIN final_filter),

all_sides AS (SELECT competitor_name, side, match_id FROM
all_competitors NATURAL JOIN current_competitors),

bet_data AS (SELECT * FROM bet NATURAL JOIN final_filter),

old_odds AS (SELECT match_id, bet_type, MAX(change_date) AS
change_date FROM (SELECT * FROM bet NATURAL JOIN final_filter
WHERE active = 'false') AS inactive_bets GROUP BY match_id, bet_type)

SELECT * FROM match_data NATURAL JOIN bet_data NATURAL JOIN
all_sides NATURAL JOIN old_odds
```

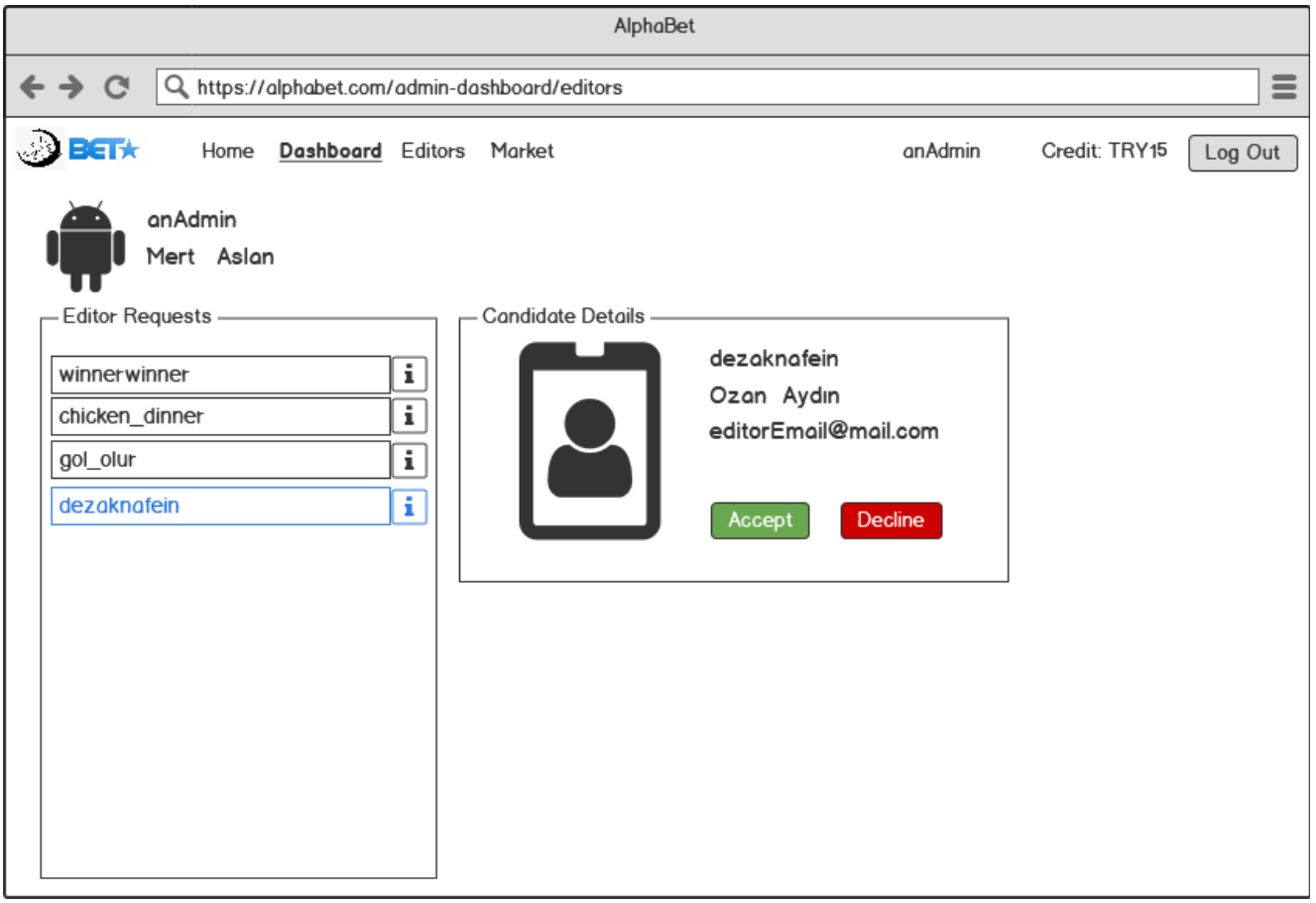
Change the odd of a specific bet:

```
UPDATE bet
SET active = ‘false’
WHERE bet_id = @bet_id AND match_id = @match_id
```

INSERT INTO bet (match_id, bet_type, change_date, odd, mbn, result, active) VALUES (@match_id, @bet_type, NOW(), @new_odd_value, @mbn, ‘pending’, ‘true’)

Cancel a specific bet:
UPDATE bet
SET active = ‘false’
WHERE bet_id = @bet_id AND match_id = @match_id

3.7.2. Admin Editor Requests Page



Inputs: @admin_id, @username

Process: In this page, the admin evaluates the editor application and either chooses to accept or decline an application. When a user selects the editor while signing up, an application will be formed and sent to the admin dashboard. By clicking on the information logo next to the name, the input @username will be sent to the backend, the admin can see the user name, real name and the e-mail of the user that applied to be an editor. The admin can now either accept the application and allow this user to sign up as an editor, or decline the application and this user has to

SQL Statements:

Display editor requests:

SELECT username FROM approves AS a INNER JOIN person AS p ON a.editor_id = p.person_id WHERE state = 'PENDING'

Display editor information:

SELECT username, forename, surname, email FROM approves AS a INNER JOIN person AS p ON a.editor_id = p.person_id WHERE state = 'PENDING'

Accept editor application:

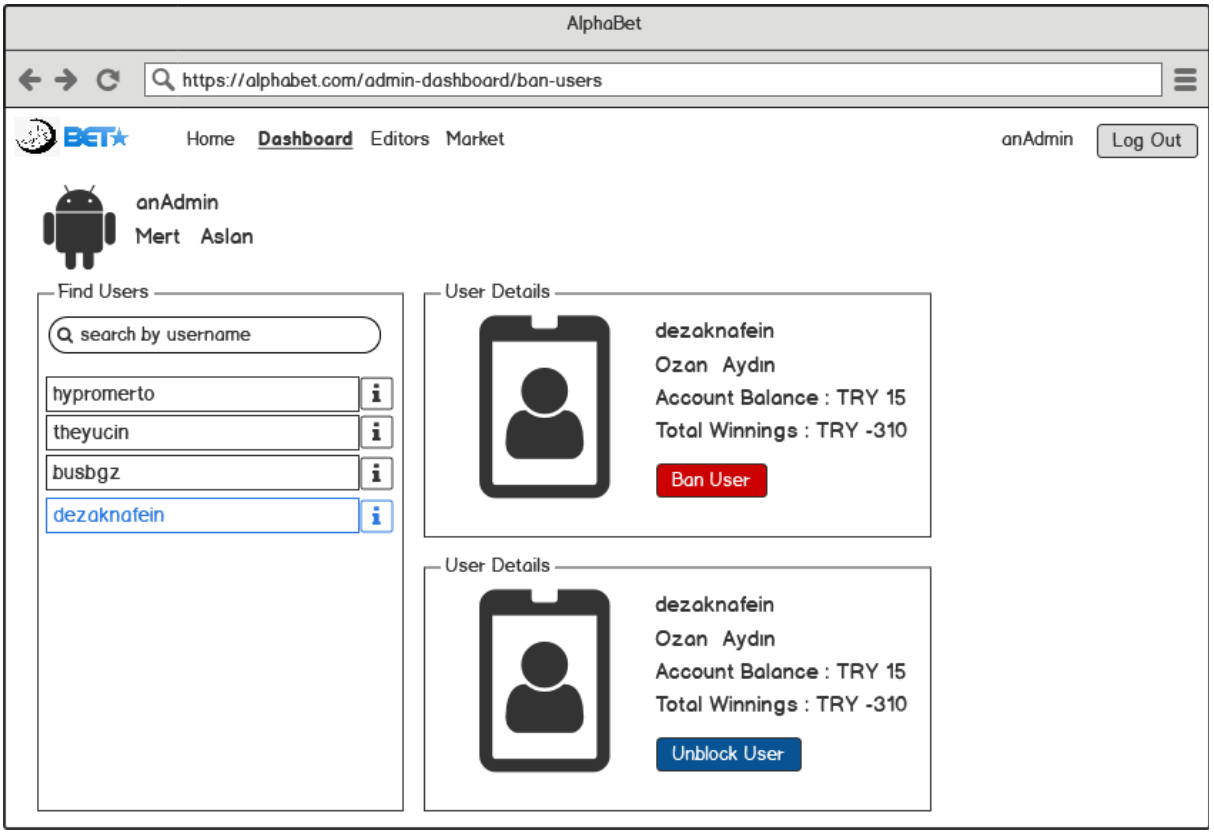
INSERT INTO editor(editor_id, win_rate, total_winnings) VALUES((SELECT person_id FROM person WHERE username = @username), 0, 0)

UPDATE approves SET state = 'APPROVED' WHERE editor_id = (SELECT person_id FROM person WHERE username = @username)

Decline editor application:

DELETE FROM person WHERE username = @username

3.7.3. Admin Ban Page



Inputs: @admin_id, @search_username, @username

Process: This page will include a text based search that the admin will use to find a specific user. By clicking on the information icon next to the person's name, firstly it will send the input @username to the backend and the admin can view that specific person's information. Afterwards, the admin will be able to ban that user if the user is an active user, or unban that user if the user is already banned.

SQL Statements:

Search users by username:

SELECT username FROM user AS u INNER JOIN person AS p ON u.user_id = p.person_id WHERE username LIKE "%@search_username%"

Display details of user:

SELECT username, forename, surname, account_balance, total_winnings FROM user AS u INNER JOIN person AS p ON user_id = person_id WHERE username = @username

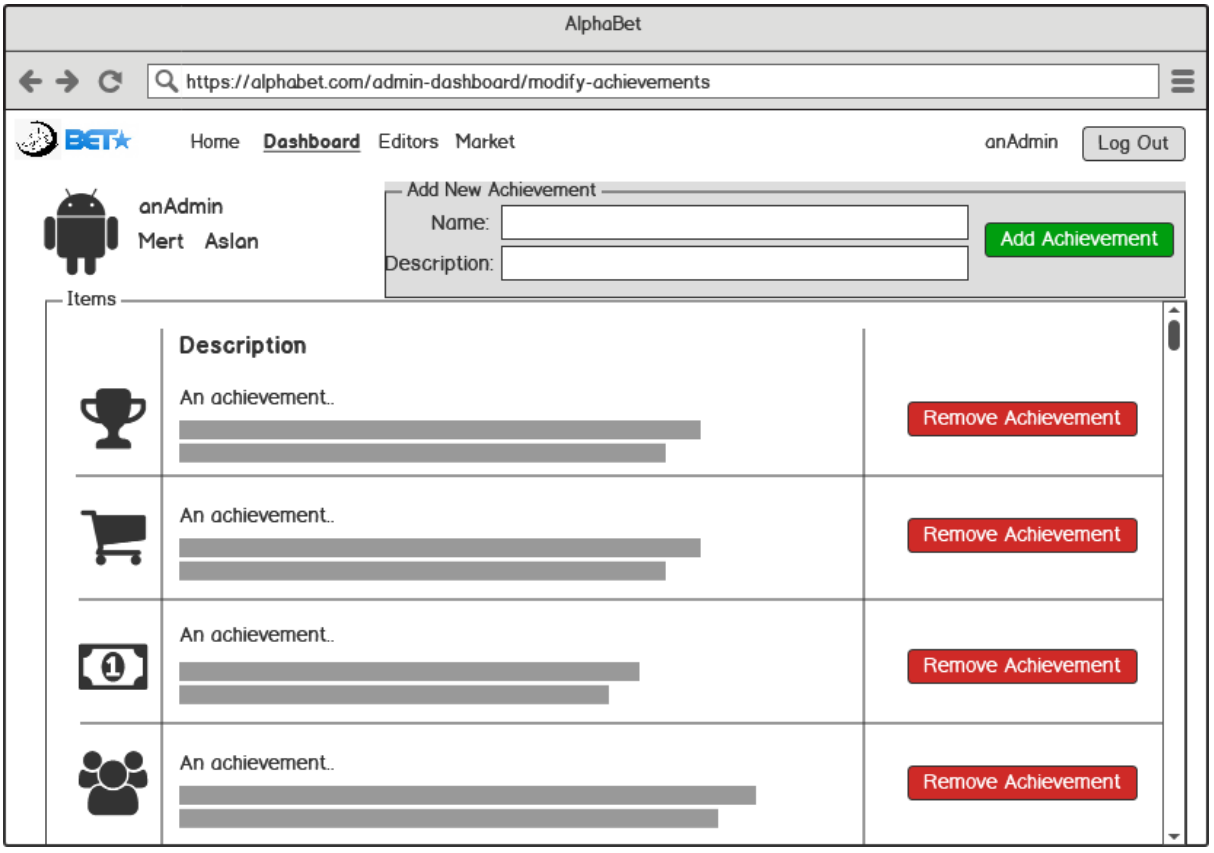
Ban user:

INSERT INTO bans(user_id, admin_id) VALUES((SELECT person_id FROM person WHERE username = @username), @admin_id)

Unblock user:

DELETE FROM bans WHERE user_id = (SELECT person_id FROM person WHERE username = @username)

3.7.4. Admin Modify Achievements Page



Inputs: @admin_id, @achievement_name, @achievement_description, @deleted_achievement_id

Process: On this page, admins can modify and add an achievement to the market. In order to add a new achievement, admins must enter achievement name and achievement description, and then click on the "Add New Achievement" button. Admins can also remove existing achievements.

SQL Statements:

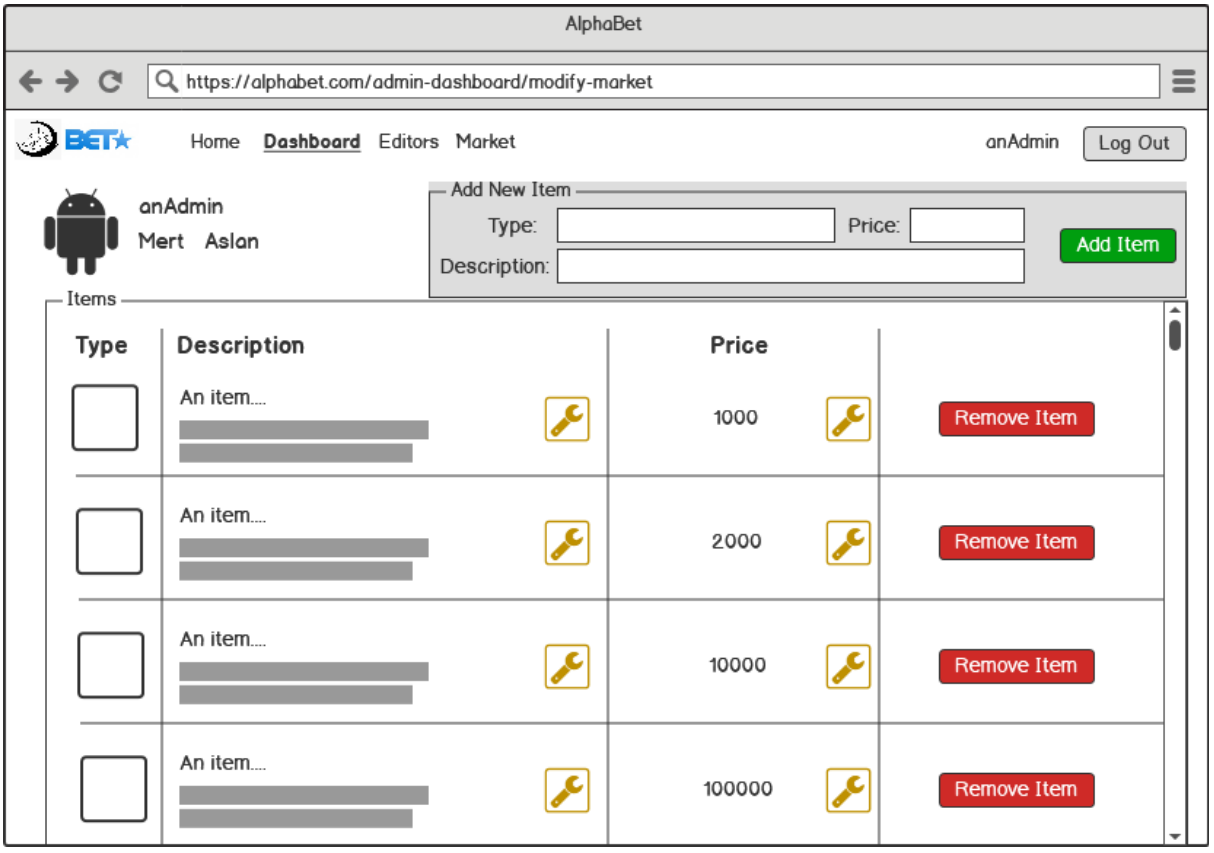
Add an achievement to achievements:

```
INSERT INTO achievement(achievement_name, achievement_description)
                VALUES(@achievement_name, @achievement_description)

DECLARE @inserted_achievement_id INT
SET @inserted_achievement_id = SCOPE_IDENTITY()

INSERT INTO added_achievement(admin_id, achievement_id)
                VALUES(@admin_id, @inserted_achievement_id)
```

Remove an achievement from market:

```
DELETE FROM achievement WHERE achievement_id =
@deleted_achievement_id
```

3.7.5. Admin Modify Market Page



Inputs: @admin_id, @new_item_type, @new_item_description, @new_item_cost, @selected_item_id, @updated_cost, @updated_description, @item_type

Process: On this page, admins can modify and add an item to the market. In order to add a new item, admins must enter item type, item description and item price, and then click on the “Add Item” button. Admins can also modify the price and description of the selected item, and can completely remove the item by clicking on the “Remove Item” button. The listing of all market items are done exactly like the market place screen.

SQL Statements:

Add an item to market:

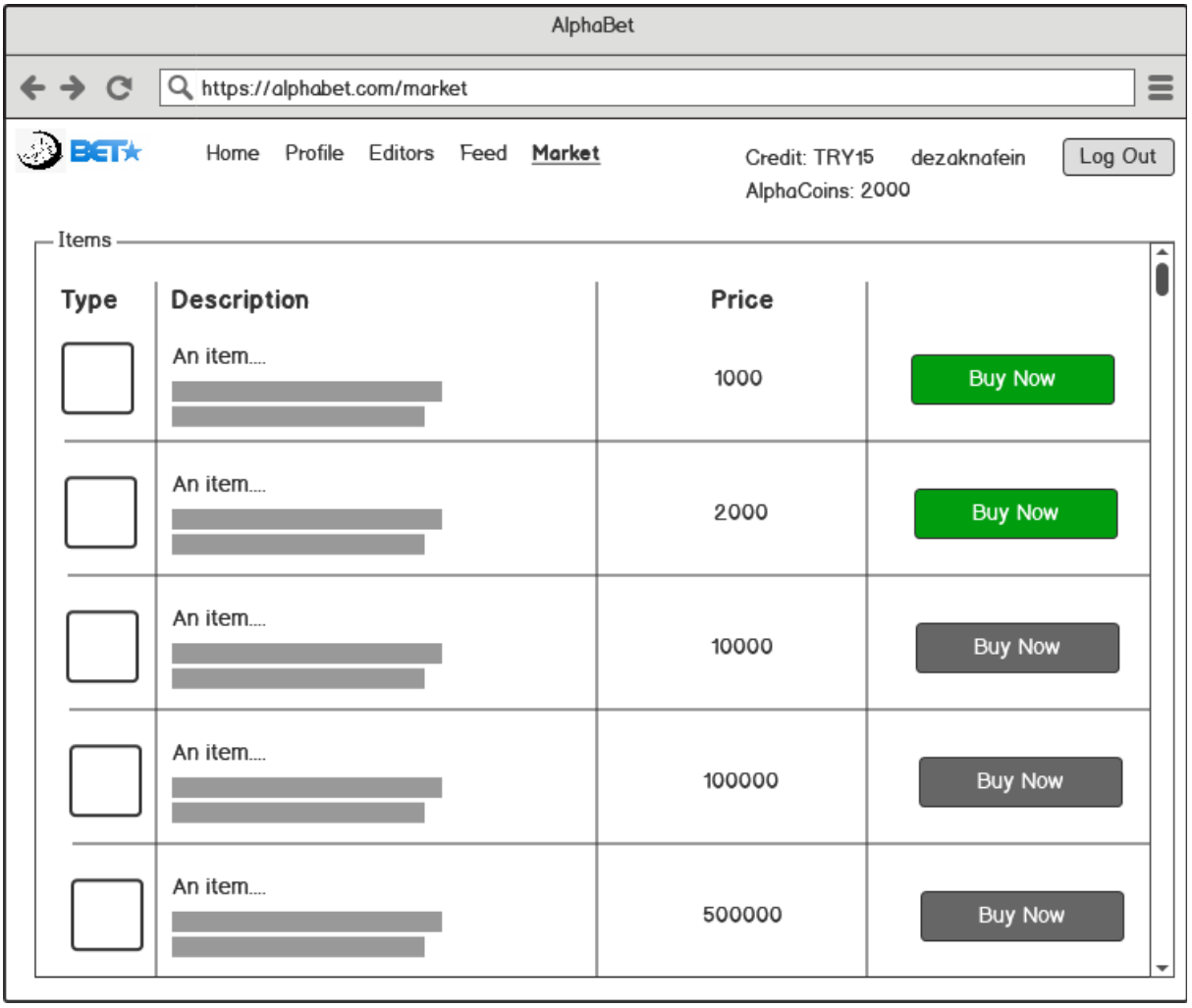
```
INSERT INTO shop_item(item_type, item_description, cost)
                VALUES(@new_item_type, @new_item_description,
        @new_item_cost)

DECLARE @inserted_item_id INT
SET @inserted_item_id = SCOPE_IDENTITY()

INSERT INTO added_item(admin_id, shop_item_id, item_type)
                VALUES(@admin_id, @inserted_item_id, @new_item_type)
```

Update cost of an item from market:
UPDATE shop_item
SET cost = @updated_cost
WHERE item_type = (SELECT item_type FROM shop_item WHERE shop_item_id = @selected_item_id)

Update description of an item from market:
UPDATE shop_item
SET description = @updated_description
WHERE item_type = (SELECT item_type FROM shop_item WHERE shop_item_id = @selected_item_id)

Remove an item from market:
DELETE FROM shop_item WHERE shop_item_id = @selected_item_id AND item_type = @item_type NOT EXISTS (SELECT shop_item_id, item_type FROM bought_item WHERE shop_item_id = @selected_item_id AND item_type = @item_type)

3.8. Market Page



Inputs: @item_id, @user_id, @item_key

Process: Here, users can spend their AlphaCoins on real life commodities like headphones, real life money and even cars! If the user has enough AlphaCoins, he can interact with the “Buy Now” button in order to buy that particular item.

SQL Statements:

Buy an item:

```
INSERT INTO bought_item (shop_item_id, user_id, item_key) VALUES (
@item_id, @user_id, @item_key)
```

List available items:

```
SELECT DISTINCT item_type, item_description, cost FROM shop_item
WHERE NOT EXISTS (SELECT shop_item_id FROM bought_item WHERE
shop_item_id = @item_id AND item_key = @item_key)
```

4. Implementation Plan

MySQL will be used for database management because it is easy to learn and well supported. Front-end will be implemented with HTML, CSS, Javascript, using ReactJS framework. Back-end will be implemented with Python, using the Flask framework. Cerberus will be used for validation.

5. Website

Our project information website link: https://busrabgz.github.io/