İHSAN DOĞRAMACI BİLKENT UNİVERSİTY

SPRING 2018

CS353 - DATABASE SYSTEMS

TERM PROJECT PROPOSAL REPORT

Social Gaming Marketplace - Ethereal

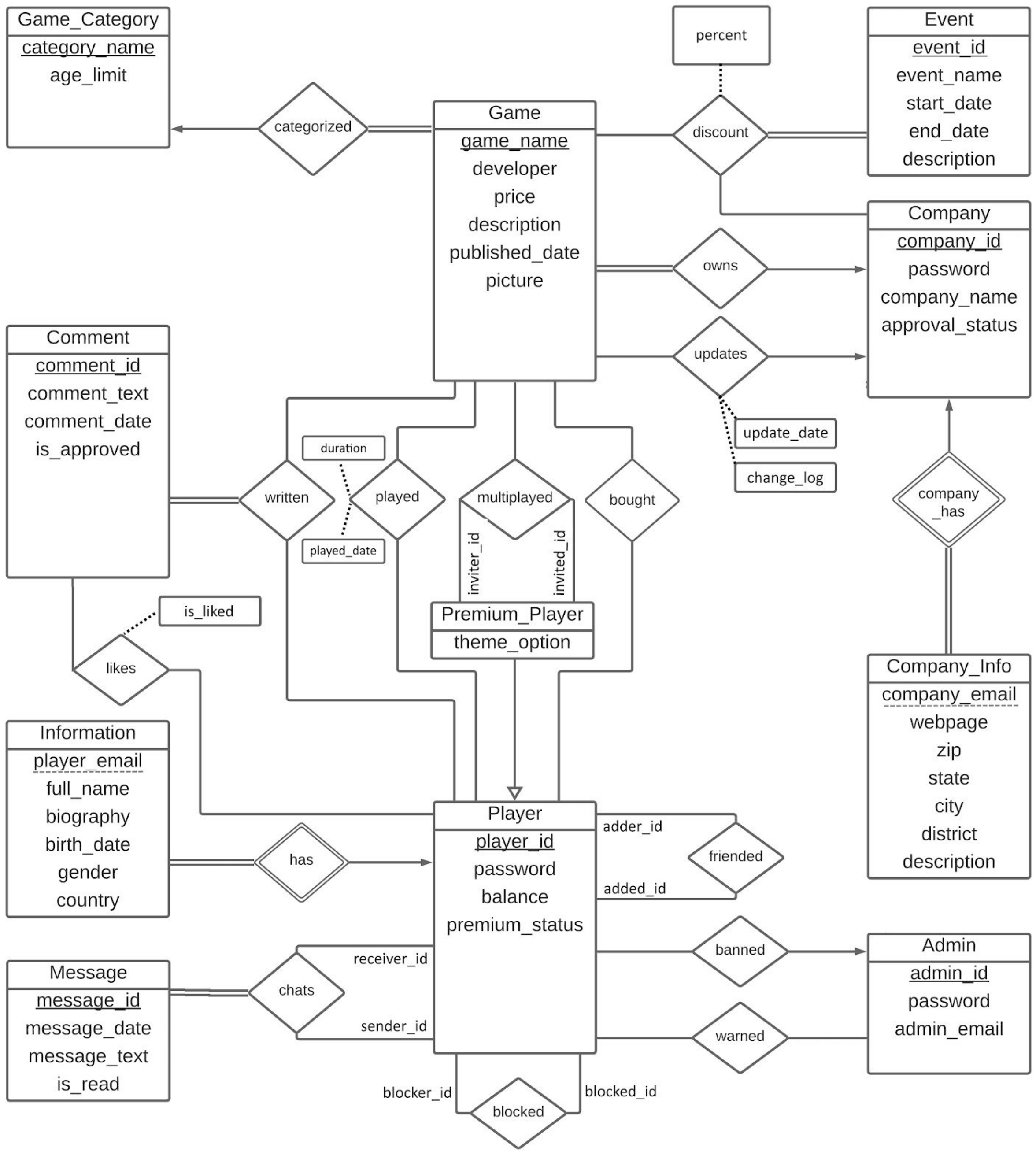
GROUP 23

GROUP MEMBERS:

1. Bikem Çamli, 21501492, Section 2
2. Mert Osman Dönmezyürek, 21301890, Section 2
3. Nursena Kal, 21501322, Section 1
4. Mahin Khankishizade, 21503495, Section 1

Table of Contents

1. Revised E/R Model



1.1 The changes made in the ER diagram

- Relation named Written was updated into a strong many to many relation between Game and Player. Comment now has a total participation in this relation.
- Relation named Discount was updated into a strong relation from a weak one. This relation keeps attribute “percent”. Event now has a total participation in this relation.
- Relation named Chats was updated into a strong relation from weak one. It is now many to many relation between Player and Player. Message has a total participation in this relation.
- Relation named Warned was added between Admin and Player. It is a many to many relation.
- New table – Company_Info was added. This table is a weak entity and keeps basic information about the company inside.
- New relation named Company_Has was added between Company and Company_Info. Company_Info has total participation in this relation.
- Relation named Updates was added between Company and Game. It has “update_date” and “changed_log” attributes in it. Updates is a one to many relation.
- Relation Played and entity Played_History was deleted. Plays relation is renamed to Played and has attributes “played_date” and “duration”.
- Player attribute “status” was renamed into “premium_status”. It checks whether the player updated their profile to premium or not.
- Dashed attributes except “player_email” in Information entity were updated into undashed attributes.
- Dashed attributes except “event_id” in Event entity were updated into undashed attributes.
- Dashed attributes except “comment_id” in Comment entity were updated into undashed attributes.
- Dashed attributes except “message_id” in Message entity were updated into undashed attributes.
- New relation added between Premium_Player and Game named “invitation”.

2 Relation Schema

2.1 Game

Relational Model:

Game(game_name, developer, description, published_date, picture, price)

Functional Dependencies:

game_name-> developer, description, published_date, picture, price

Keys
Candidate Keys { (game_name) }
Primary key: game_name
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Game(
        game_name        varchar(20) primary key,
        developer        varchar(20) not null,
        description      varchar(150),
        published_date   date not null,
        picture          varchar(200),
        price            float(10) not null)
```

2.2 Game_Category

Relational Model:
Game_Category(category_name, age_limit)

Functional Dependencies:
category_name -> age_limit

Keys
Candidate Keys { (category_name) }
Primary key: category_name
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Game_Category(
        category_name    varchar(20) primary key,
        age_limit        int(2) not null)
```

2.3 Company

Relational Model:
Company(company_id, company_name, password, approval_status)

Functional Dependencies:
company_id -> company_name, password, approval_status

company_name -> company_id, password, approval_status

Keys
Candidate Keys { (company_id) }, { (company_name) }
Primary key: company_id
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Company(
        company_id      varchar(20) primary key,
        password        varchar(8) not null,
        company_name  varchar(20) not null,
                approval_status         boolean not null)
```

2.4 Event
Relational Model:
Event(<u>event_id</u>, event_name, start_date, end_date, description)

Functional Dependencies:
event_id -> event_name, start_date, end_date, description

Keys
Candidate Keys { (event_id) }
Primary key: event_id
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Event(
        event_id          varchar(20) primary key,
        event_name      varchar(20) not null,
        start_date        date not null,
        end_date          date not null,
        description      varchar(150))
```

2.5 Player
Relational Model:
Player(<u>player_id</u>, password, balance, premium_status)

Functional Dependencies:
player_id -> password, balance, premium_status

Keys
Candidate Keys { (player_id) }
Primary key: player_id
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Player(
        player_id          varchar(20)     primary key,
        password           varchar(8) not null,
        balance            float(10),
        premium_status  boolean)
```

2.6 Premium_Player
Relational Model:
Premium_Player(player_id, theme_option)

Functional Dependencies:
player_id -> theme_option

Keys
Candidate Keys { (player_id) }
Primary key: player_id
Foreign Key: None

Normal Forms: BCNF
Table Definition:
```
CREATE TABLE Premium_Player(
        player_id          varchar(20),
        theme_option     varchar(20),
                foreign key (player_id) references Player(player_id))
```

2.7 Admin
Relational Model:
Admin(admin_id, password, admin_email)

Functional Dependencies:
admin_id -> password, admin_email

Keys
Candidate Keys { (admin_id) }, { (admin_email) }
Primary key: admin_id
Foreign Key: None

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Admin(
        admin_id        varchar(20) primary key,
        password        varchar(8) not null,
        admin_email     varchar(20) not null)
```

2.8 Information

Relational Model:
Information(player_id, player_email, full_name, birth_date, gender, country, biography)
FK: player_id references Player(player_id)

Functional Dependencies:
player_id, player_email -> full_name, birth_date, gender, country, biography

Keys
Candidate Keys { (player_id, player_email) }
Primary key: player_id, player_email
Foreign Key: player_id references Player

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Information(
        player_email    varchar(20),
        player_id       varchar(20),
        full_name       varchar(40),
        birth_date      date,
        gender          varchar(20),
        country         varchar(20),
        biography       varchar(200),
        primary key( player_email, player_id),
        foreign key(player_id) references Player(player_id))
```

2.9 Message

Relational Model:

Message(message_id, message_date, message_text)

Functional Dependencies:

message_id -> message_date, message_text

Keys

Candidate Keys { (message_id) }

Primary key: message_id

Foreign Key: None

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Message(
        message_id        varchar(20) primary key,
        message_date      date not null,
        message_text      varchar(200) not null)
```

2.10 Comment

Relational Model:

Comment(comment_id, comment_text, comment_date, like_count, dislike_count, approval_status)

Functional Dependencies:

comment_id -> comment_text, comment_date, like_count, dislike_count, approval_status

Keys

Candidate Keys { ((comment_id)) }

Primary key: comment_id

Foreign Key: None

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Comment(
        comment_id        varchar(20) primary key,
        approval_status           boolean not null,
        comment_text      varchar(200) not null,
        comment_date      date not null,
        like_count        int(10),
        dislike_count       int(10))
```

2.10 Company_Info

Relational Model:

Company_Info(company_id, company_email, webpage, zip, state, city, district, description_info)

FK: company_id references Company(company_id)

Functional Dependencies:

company_id, company_email -> webpage, zip, state, city, district, description_info
zip -> state, city, district

Keys

Candidate Keys { (company_id, company_email) }
Primary key: company_id, company_email
Foreign Key: copmany_id references Company

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Company_Info(
                webpage          varchar(20),
                company_id       varchar(20),
                copmany_email    varchar(20),
        zip                 char(6),
        state               varchar(20),
        city                varchar(20),
        district            varchar(20),
        description         varchar(200),
        primary key (company_id, company_email),
        foreign key company_id references Company(company_id))
```

2.11 Categorized

Relational Model:

Categorized(category_name, game_name)

FK: category_name references Game_Category(category_name)
FK: game_name references Game(game_name)

Functional Dependencies:

None

Keys

Candidate Keys { (category_name, game_name) }
Primary key: category_name, game_name
Foreign Key: category_name references Game_Category, game_name references Game

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Categorized(
        category_name   varchar(20),
        game_name       varchar(20),
        primary key( category_name, game_name),
                foreign key (category_name) references Game_Category(category_name),
                foreign key (game_name) references Game(game_name))
```

2.12 Owns
Relational Model:
Owns(game_name, company_id)
FK: company_id references Company(company_id)
FK: game_name references Game(game_name)

Functional Dependencies:
None

Keys
Candidate Keys { (game_name, company_id) }
Primary key: game_name, company_id
Foreign Key: game_name references Game, company_id references Company

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Owns(
        company_id      varchar(20),
        game_name       varchar(20),
        primary key( company_id, game_name),
                foreign key (company_id) references Company(company_id),
                foreign key (game_name) references Game(game_name))
```

2.13 Updates
Relational Model:
Updates(game_name, company_id, last_date, change_log)
FK: company_id references Company(company_id)

FK: game_name references Game(game_name)

Functional Dependencies:
None

Keys
Candidate Keys { (game_name, company_id) }
Primary key: game_name, company_id
Foreign Key: game_name references Game, company_id references Company

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Updates(
        company_id       varchar(20),
        game_name        varchar(20),
        update_date      date not null,
        change_log       varchar(200) not null,
        primary key( company_id, game_name),
        foreign key (company_id) references Company(company_id),
                foreign key (game_name) references Game(game_name))
```

2.14 Discount

Relational Model:
Discount (<u>game_name, company_id, event_id,</u> percent)

Functional Dependencies:
None

Keys
Candidate Keys { (game_name, company_id, event_id) }
Primary key: game_name, company_id, event_id
Foreign Key: game_name references Game,
company_id references Company,
event_id references Event
Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Discount(
        company_id       varchar(20),
        game_name        varchar(20),
        event_id         varchar(20),
        percent             int(2) not null,
        primary key( company_id, game_name, event_id),
        foreign key (company_id) references Company(company_id),
        foreign key( game_name) references Game(game_name),
        foreign key(event_id) references Event(event_id))
```

2.15 Written

Relational Model:

Written(game_name, player_id, comment_id)

FK: game_name references Game(game_name)

FK: player_id references Player(player_id)

FK: comment_id references Comment(comment_id)

Functional Dependencies:

None

Keys

Candidate Keys { (game_name, player_id, comment_id) }

Primary key: game_name, player_id, comment_id

Foreign Key: game_name references Game,

comment_id references Comment,

player_id references Player

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Written(
        comment_id      varchar(20),
        player_id       varchar(20),
        game_name        varchar(20),
                primary key(comment_id, player_id, game_name),
                foreign key (comment_id) references Comment(comment_id),
                foreign key (player_id) references Player(player_id),
                foreign key (game_name) references Game(game_name))
```

2.16 Bought

Relational Model:

Bought(game_name, player_id)

FK: game_name references Game(game_name)

FK: player_id references Player(player_id)

Functional Dependencies:

None

Keys
Candidate Keys { (game_name, player_id) }
Primary key: game_name, player_id
Foreign Key: game_name references Game,
player_id references Player

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Bought(
        player_id          varchar(20),
        game_name        varchar(20),
                primary key( player_id, game_name),
                foreign key (player_id) references Player(player_id),
                foreign key (game_name) references Game(game_name))
```

2.17 Played

Relational Model:
Played(<u>game_name, player_id,</u> duration, played_date)
FK: game_name references Game(game_name)
FK: player_id references Player(player_id)

Functional Dependencies:
None

Keys
Candidate Keys { (game_name, player_id) }
Primary key: game_name, player_id
Foreign Key: game_name references Game,
player_id references Player

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Played(
        player_id          varchar(20),
        game_name          varchar(20),
        duration           int(5) not null,
        played_date        date not null,
        primary key( game_name, player_id),
        foreign key (game_name) references Game(game_name),
        foreign key (player_id) references Player(player_id))
```

2.18 Has

Relational Model:

Has(<u>player_id, player_email</u>)

FK: player_id references Player(player_id)
FK: player_email references Information(player_email)

Functional Dependencies:

None

Keys

Candidate Keys { (player_id, player_email) }
Primary key: player_id, player_email
Foreign Key: player_id references Player,
player_email references Information

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Has(
        player_id           varchar(20),
        player_email        varchar(20),
        primary key( player_id, player_email),
        foreign key (player_id) references Player(player_id),
        foreign key (player_email) references Information(player_email))
```

2.19 Chats

Relational Model:

Chats(<u>sender_id, receiver_id, message_id</u>)

FK: sender_id references Player(player_id)
FK: receiver_id references Player(player_id)
FK: message_id references Message(message_id)

Functional Dependencies:

None

Keys

Candidate Keys { (sender_id, receiver_id, message_id) }
Primary key: sender_id, receiver_id, message_id
Foreign Key: sender_id reference to Player,
receiver_id reference to Player,
message_id reference to Message

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Chats(
        sender_id           varchar(20),
        receiver_id         varchar(20),
        message_id          varchar(20),
        primary key( sender_id, receiver_id, message_id),
        foreign key (sender_id) references Player(player_id),
        foreign key (receiver_id) references Player(player_id),
        foreign key (message_id) references Message(message_id))
```

2.20 Blocked

Relational Model:

Blocked(<u>blocker_id, blocked_id</u>)

FK: blocker_id references Player(player_id)

FK: blocked_id references Player(player_id)

Functional Dependencies:

None

Keys

Candidate Keys { (blocker_id, blocked_id) }

Primary key: blocker_id, blocked_id

Foreign Key: blocker_id reference to Player,
blocked_id reference to Player

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Blocked(
        blocker_id          varchar(20),
        blocked_id          varchar(20),
        primary key( blocker_id, blocked_id),
        foreign key (blocker_id) references Player(player_id),
        foreign key (blocked_id) references Player(player_id))
```

2.21 Friended

Relational Model:

Friended(<u>adder_id, added_id</u>)

FK: adder_id references Player(player_id)

FK: added_id references Player(player_id)

Functional Dependencies:
None

Keys
Candidate Keys { (adder_id, added_id) }
Primary key: adder_id, added_id
Foreign Key: added_id references Player,
adder_id references Player

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Friended(
        adder_id            varchar(20),
        added_id            varchar(20),
        primary key(adder_id, added_id),
        foreign key (adder_id) references Player(player_id),
        foreign key (added_id) references Player(player_id))
```

2.22 Banned

Relational Model:
Banned(<u>player_id, admin_id</u>)
FK: player_id references Player(player_id)
FK: admin_id references Admin(admin_id)

Functional Dependencies:
None

Keys
Candidate Keys { (player_id, admin_id) }
Primary key: player_id, admin_id
Foreign Key: player_id reference to Player,
admin_id reference to Admin

Normal Forms: BCNF

Table Definition:
```
 CREATE TABLE Banned(
        player_id           varchar(20),
        admin_id            varchar(20),
        primary key(player_id, admin_id),
        foreign key (player_id) references Player(player_id),
                    foreign key (admin_id) references Admin(admin_id))
```

2.23 Warned

Relational Model:

Warned(player_id ,admin_id)
FK: player_id references Player(player_id)
FK: admin_id references Admin(admin_id)

Functional Dependencies:

None

Keys

Candidate Keys { (player_id, admin_id) }
Primary key: player_id, admin_id
Foreign Key: player_id references Player,
admin_id references Admin

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Warned(
        player_id           varchar(20),
        admin_id           varchar(20),
        primary key( player_id, admin_id),
        foreign key (player_id) references Player(player_id),
        foreign key (admin_id) references Admin(admin_id))
```

2.24 Approves

Relational Model:

Approves(company_id, admin_id, status)
FK: company_id references Company(company_id)
FK: admin_id references Admin(admin_id)

Functional Dependencies:

Nope

Keys

Candidate Keys { (company_id, admin_id) }
Primary key: company_id, admin_id
Foreign Key: company_id references Company,
admin_id references Admin

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Approves(
          company_id       varchar(20),
          admin_id           varchar(20),
         status                boolean not null,
                    primary key (company_id, admin_id),
                    foreign key (company_id) references Company(company_id),
                    foreign key (admin_id) references Admin(admin_id))
```

2.25 Likes

Relational Model:

Likes(<u>player_id, comment_id</u>, is_liked)

FK: player_id references Player(player_id)

FK: comment_id references Comment(comment_id)

Functional Dependencies:

Nope

Keys

Candidate Keys { (player_id, comment_id) }

Primary key: player_id, comment_id

Foreign Key: player_id references Player,
comment_id references Comment

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE Likes(
          player_id        varchar(20),
                    comment_id    varchar(20),
                    is_liked           boolean,
                    primary key (player_id, comment_id),
                    foreign key (player_id) references Player(player_id),
                    foreign key (comment_id) references Comment(comment_id))
```

2.26 Company_Has

Relational Model:

Company_Has(<u>company_id, company_email</u>)

FK: company_id references Company(company_id)

FK: company_email references Company_Info(company_email)

Functional Dependencies:
Nope

Keys
Candidate Keys { (company_id, company_email) }
Primary key: company_id, company_email
Foreign Key: company_id references Company,
company_email references Company_Info

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Company_Has(
        company_id              varchar(20),
        company_email           varchar(20),
        primary key (company_id, company_email),
        foreign key (company_id) references Company(company_id),
        foreign key (company_email) references Company_Info(company_email))
```

2.27 Multiplayed

Relational Model:
Multiplayed(<u>inviter_id, invited_id, game_name</u>)
FK: invited_id references Player(player_id)
FK: inviter_id references Player(player_id)
FK: game_name references Game(game_name)

Functional Dependencies:
Nope

Keys
Candidate Keys { ((inviter_id, invited_id, game_name)) }
Primary key: inviter_id, invited_id, game_name
Foreign Key: inviter_id references Player,
invited_id references Player,
game_name references Game

Normal Forms: BCNF

Table Definition:
```
CREATE TABLE Multiplayed(
        inviter_id      varchar(20),
        invited_id      varchar(20),
        game_name       varchar(20),
        primary key (inviter_id, invited_id, game_name),
        foreign key (inviter_id) references Player(player_id),
        foreign key (invited_id) references Player(player_id),
        foreign key (game_name) references Player(game_name))
```

3. Functional Dependencies and Normalization of Tables

We specify all of the functional dependencies and normal forms in the Relation Schema part (Part 2). As it can be seen from the previous part, all of the relations are in Boyce-Codd Normal Form (BCNF). Thus, neither any decomposition nor normalization is required for our system.

4. Functional Components

4.1 Use Cases/Scenarios

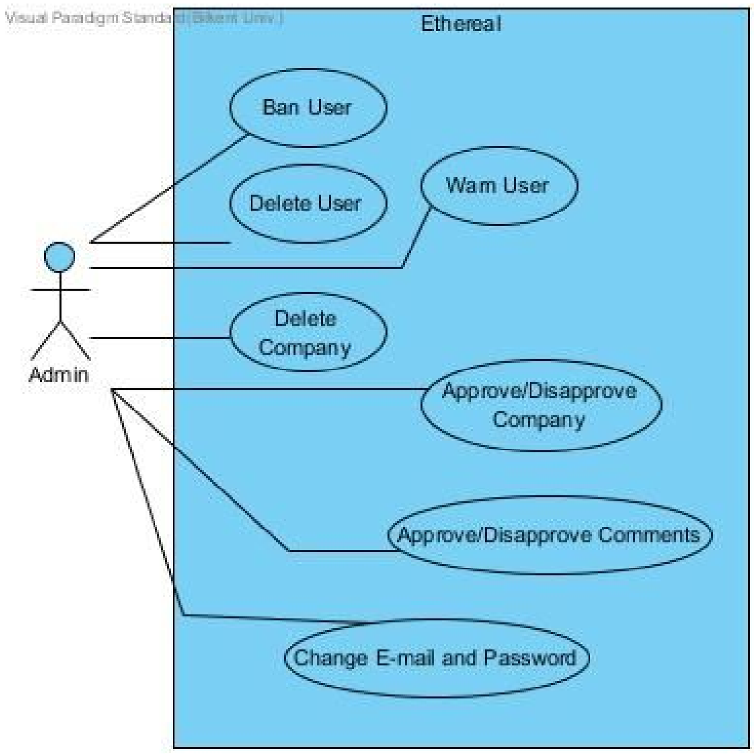
Ethereal has three different types of users. Use cases are as follows.

4.1.1 Admin

The admins are the administrators of Ethereal. This means that an admin should be able to control users, their activities and update content of the website if necessary.

Admin

- An admin can ban a user
- An admin can delete a user
- An admin can warn a user
- An admin can delete company
- An admin can approve/disapprove following content:
  - An admin can approve/disapprove the company
  - An admin can approve/disapprove the comments
- An admin can change email and password


Visual Paradigm Standard(Bilkent Univ.)
Ethereal
Ban User
Delete User
Warn User
Delete Company
Approve/Disapprove Company
Approve/Disapprove Comments
Change E-mail and Password
Admin

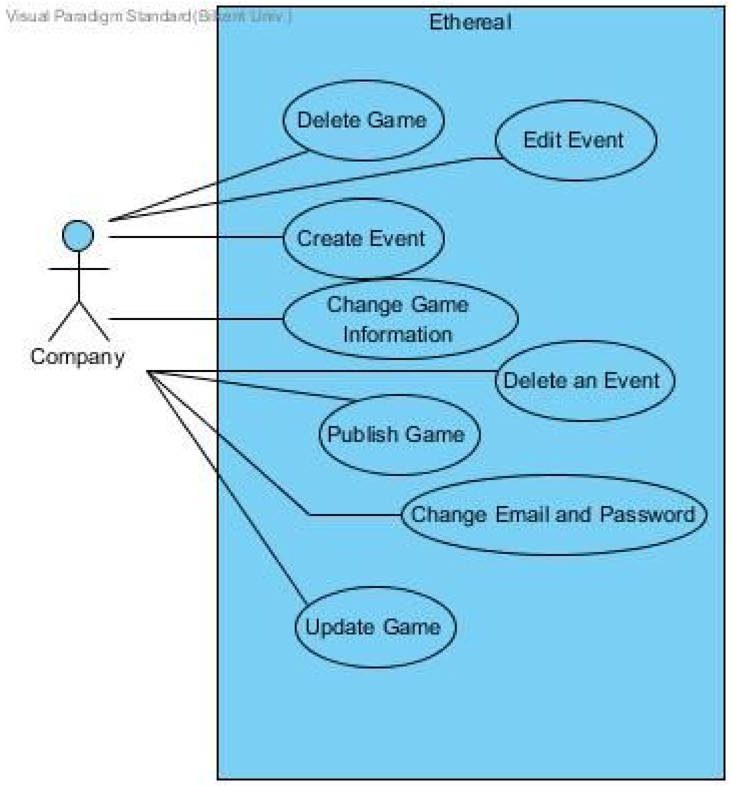
4.1.2 Company

A company is the type of account which uploads games to the system, creates events for the games and edits their information. They gain profit from their game and they can change the game information as they like.

Company

- A company can publish a game.
- A company can delete a game.
- A company can change information about the game.
- A company can update their games.
- A company can create an event
- A company can edit the existing event
- A company can delete an event

- A company can change email and password



4.1.3 Player and Premium Player

Players are the main actors of Ethereal. They can play games, buy games, chat and add their friends to the system. Also, there's a premium account type is available which adds new features to a player as being able to change themes and ability to invite friends to a game.

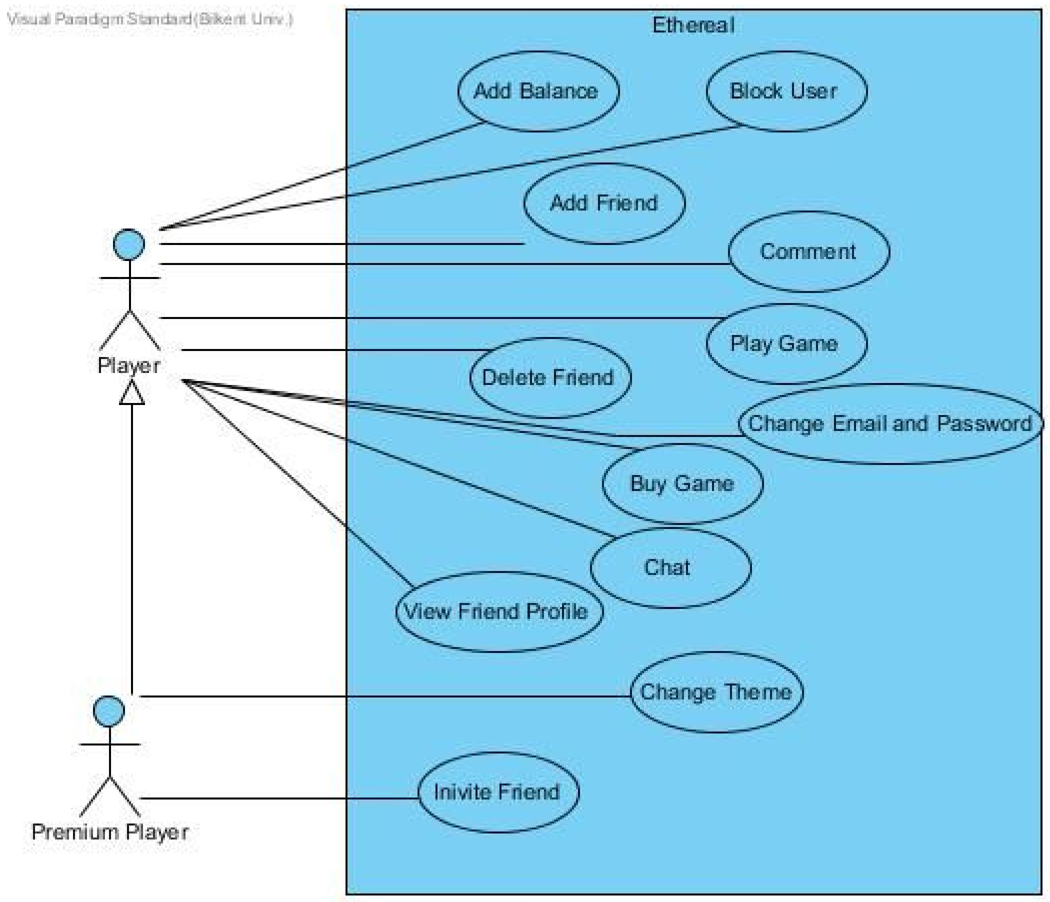
Player

- A player can add balance.
- A player can block/unblock another player.
- A player can add/unfriend a friend.
- A player can comment on games.
- A player chat with a friend.
- A player can view his/her friend’s profile.
- A player can buy a game.
- A player can play a game.
- A player can update their profile into the premium player.
- A player can change his/her email and password.

Premium Player

- A premium player can do everything that a player can do.
- A premium player can change the theme of the system.

- A premium player can invite his/her friends to a game to play together.



4.2 Algorithms

Different algorithms will be used for different types of users.

4.2.1 Player Related Algorithms

We want to keep all players' information related to the games and game categories they play. First, players will be able to see all games. By the tame, the games will get categorized by corresponding game category. Later, players will be able to see their and their friends' most played games and categories. In addition, premium players will be able to see the statistics of their invited games (i.e to which game they have been invited the most). By using the provided information, players will be able to suggest games to others or invite other players into the games that they play the most.

4.2.2 Game Related Algorithm

Since game is the main product of Ethereal, it will be kept stable and secure. The data-management system will keep track of game statistics such as the most sold/played games.

4.2.3 Logical Requirements

To prevent and minimize the logical errors some precautions will be taken.

- The date attributes of our system will be restricted to prevent possible logical mistakes. For instance, when a company publish a game or when the user share a comment, published_date and comment_date attributes of the Game and Comment tables will be filled automatically with the date of that day. In addition, when Company will create an event, chosen start_date and end_date can not be a past date. Additionally, end_date cannot be chosen earlier than start_date of the event. Therefore, start_date and end_date are boundary points in Event table and company should choose a date according to these restrictions.
- The values such as price, age_limit and balance should not be negative. By taking this precaution, the system will prevent possible consequences such as a player buys a game even if he/she does not have enough money (balance becomes negative), a player enters a game with larger age_limit restriction than his/her own age (age - age_limit should more than zero in order to enter the game). Additionally, birth_date of a player should be realistic (birth_date cannot be more than less than 1918, for example).

4.3 Data Structures

We are using built-in data types of SQL such as int, varchar, char, float, boolean, date etc. There will not be any additional data structure used.

5.User Interface Design and Corresponding SQLs

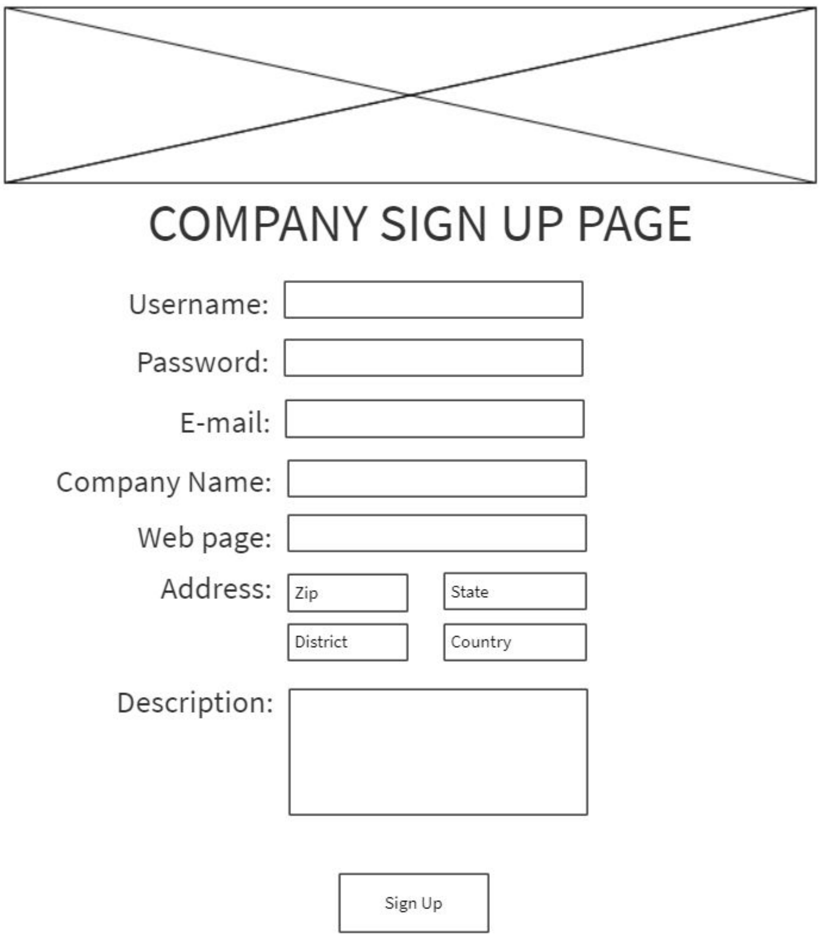
5.1 Company Sign Up Page

Inputs: @username, @password, @e-mail, @company_name, @web_page, @description, @state,@city, @zipcode, @district

Process: The company should fill all of the information on the sign-up page. The new Customer added to the database. The status of the company will be 0 in the Approves table until an admin approves the company. After the approval company can be login to the website and upload its games.

SQL Statements:

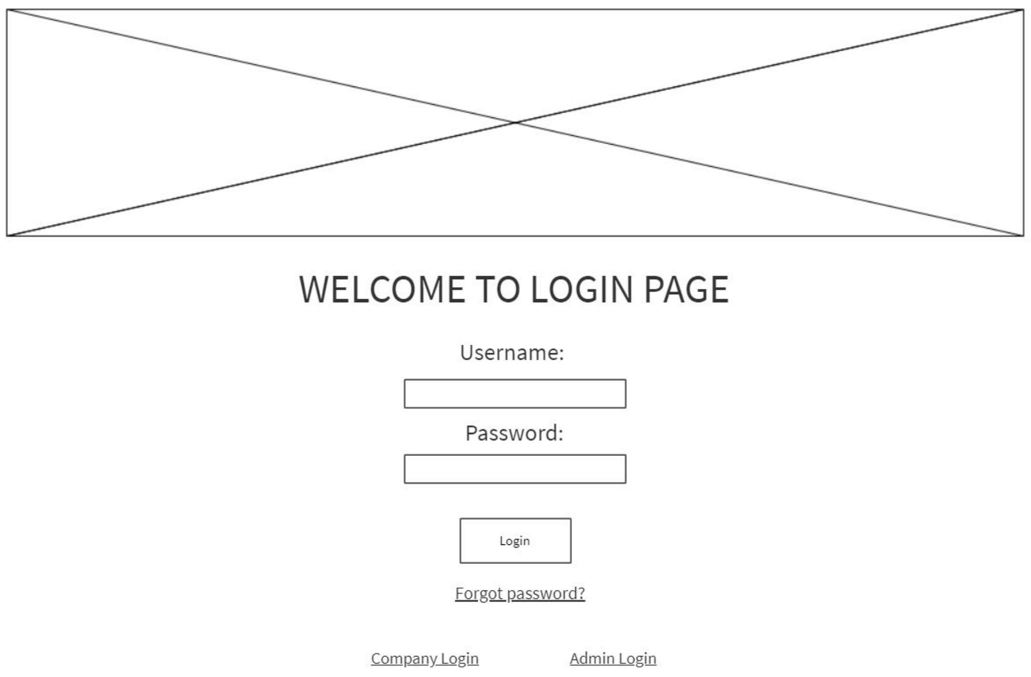
```
INSERT INTO Company VALUES( @username, @company_name, @password)
INSERT INTO Copmany_Info VALUES( @username, @e-mail, @web_page, @description)
INSERT INTO Location VALUES( @username, @zipcode, @state, @city, @district)
```



5.2 Player Log-in Page
Inputs: @username, @password

Process: Players Login with their passwords and usernames. To the players who forgot their password, a forgot password option is available.

SQL Statements:
```
SELECT player_id
FROM Player
WHERE @username = player_id AND @password = password
```


WELCOME TO LOGIN PAGE
Username:
Password:
Login
Forgot password?
Company Login
Admin Login

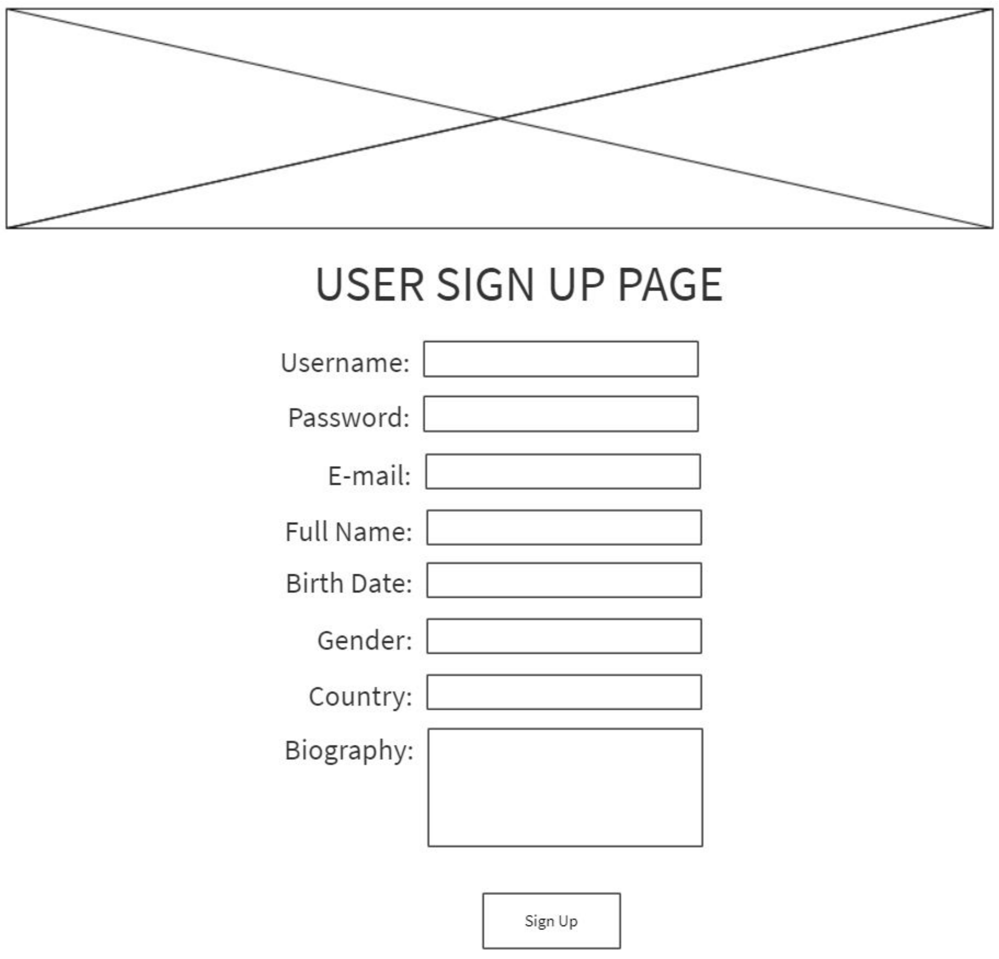
5.3 Player Sign Up Page

Inputs: @username, @password, @e-mail, @name, @surname, @birth_date, @biography, @gender,@country

Process: The player should fill all of the information on the sign-up page. The new Player is added to the database.

SQL Statements:

```
INSERT INTO Player VALUES( @username, @password)
INSERT INTO Information VALUES( @username, @e-mail, @name, @surname, @birth_date,
@gender, @country, @biography)
```



USER SIGN UP PAGE

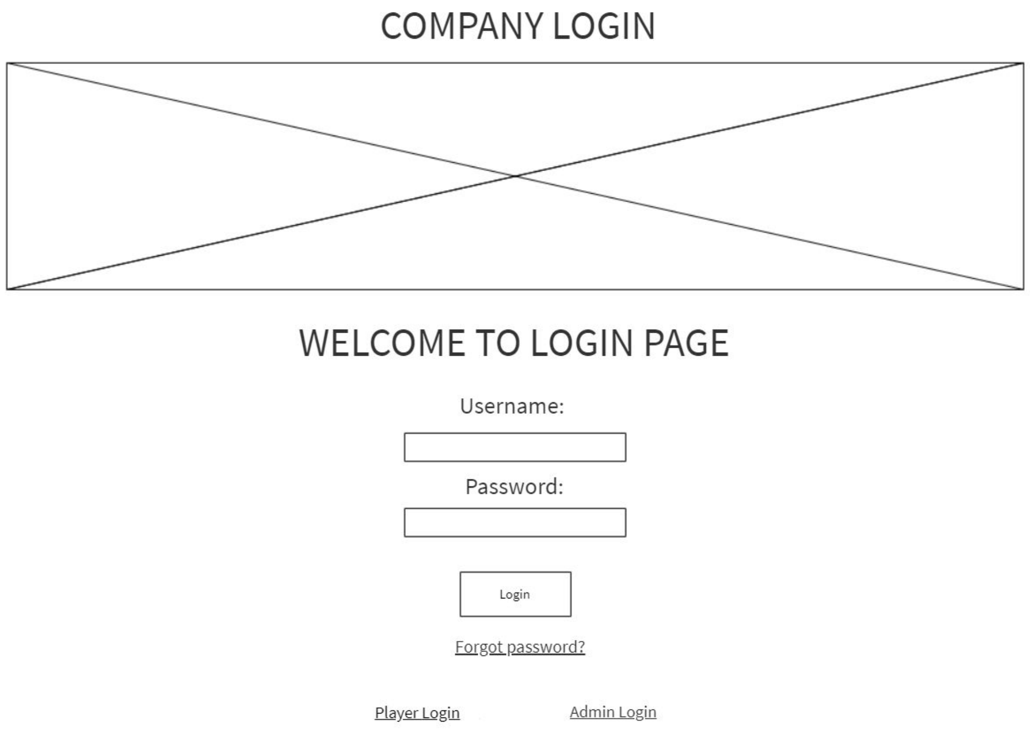
Username:	
Password:	
E-mail:	
Full Name:	
Birth Date:	
Gender:	
Country:	
Biography:	

5.4 Company Login Page
Inputs: @username, @password

Process: Companies can login with their passwords and usernames. To the companies who forgot their password, a forgot password option is available.

SQL Statements:

```
SELECT company_id
FROM Company
WHERE @username = company_id AND @password = password
```

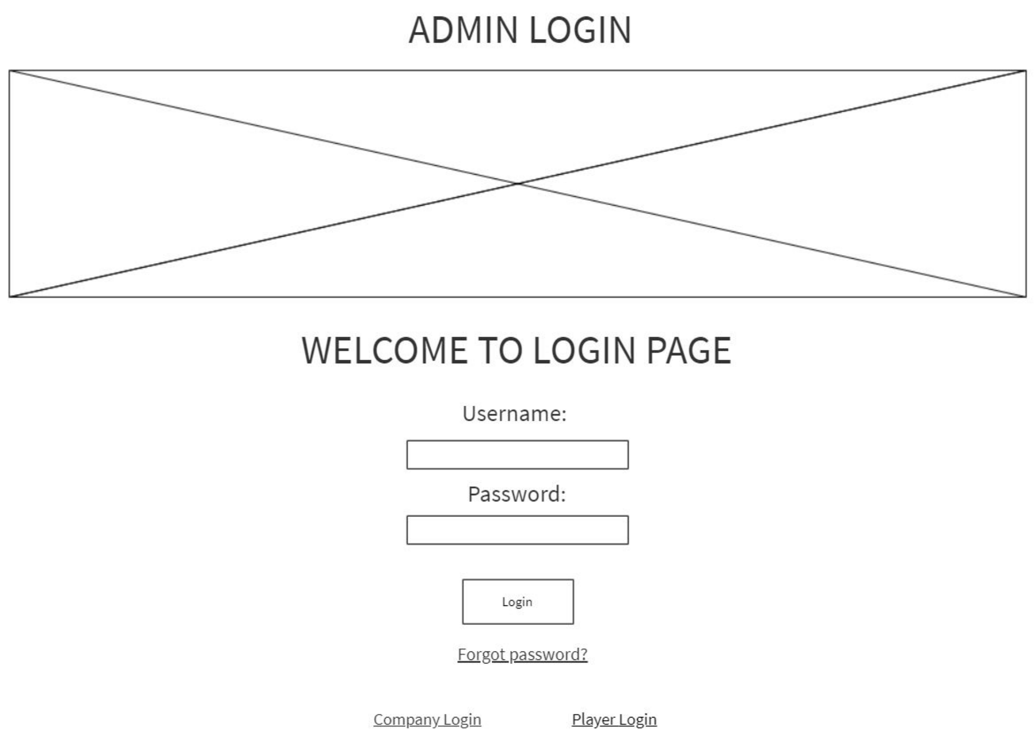


5.5 Admin Login Page
Inputs: @username, @password

Process: Admins can login with their passwords and usernames. To the admins who forgot their password, a forgot password option is available.

SQL Statements:

```
SELECT admin_id
FROM Admin
WHERE @username = admin_id AND @password = password
```



5.6 User Marketplace
Inputs: @username, @balance, @event_name, @event_description, @game_category_name, @game_name

Process: After players Login their accounts they can see the discount events of the game and see their balances, buy a new game in user marketplace page. In addition, this pages also provides players a link to the list of the games in each category, a link to a list of the games they bought and a link to go to their profiles.

SQL Statements:

Displaying the balance of the user
```
SELECT balance
FROM Player
WHERE @username = player_id
```

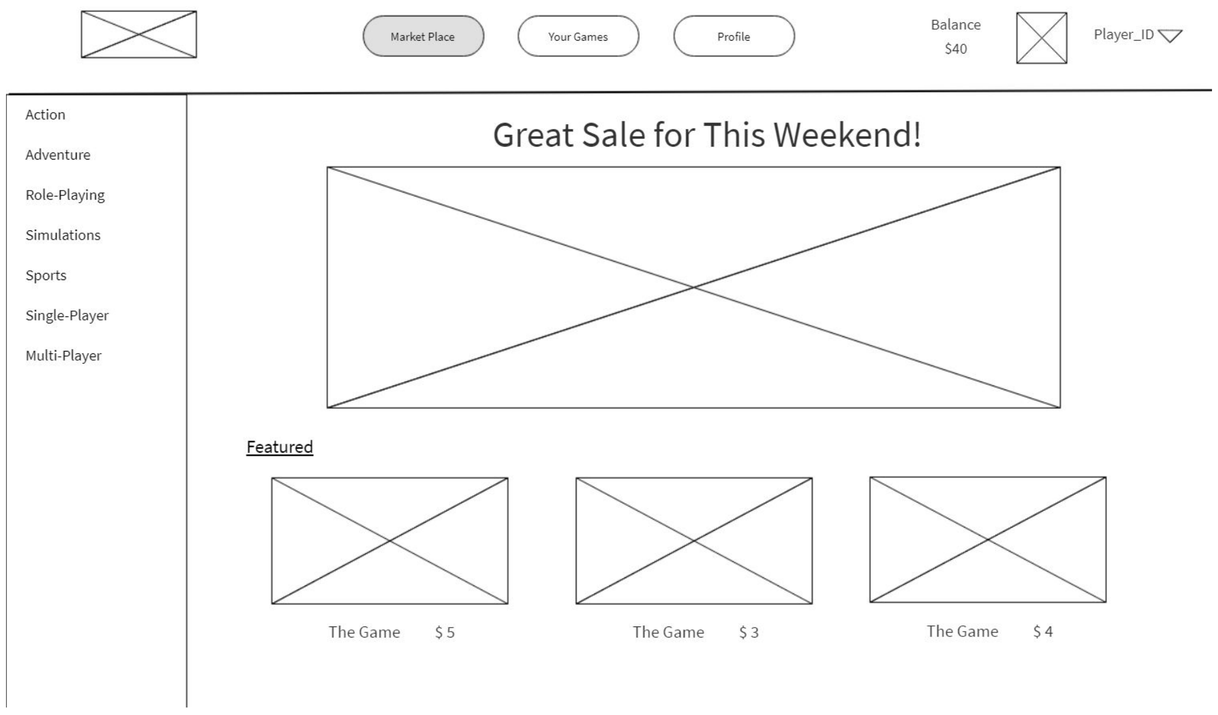
Player buys a new game
```
INSERT INTO Bought VALUES(@game_name, @player_id)
```

Displaying event names and descriptions on the screen
```
SELECT event_name, description
FROM Event
WHERE @event_name = event_name, @event_description = description
```

Displaying game categories
```
SELECT category_name
FROM Game_Category
```

Displaying the games corresponding to the pressed category name
```
SELECT game_name, developer, description, published_date, picture, price
FROM Game natural join Categorized C
WHERE @game_category_name = C.category_name
```

Displaying the list of the games the player bought
```
SELECT game_name
FROM Player natural join Bought B
WHERE @username = B.player_id
```


Market Place
Your Games
Profile
Balance
$40
Player_ID
Action
Adventure
Role-Playing
Simulations
Sports
Single-Player
Multi-Player
Great Sale for This Weekend!
Featured
The Game $ 5
The Game $ 3
The Game $ 4

5.7 Chat pop-up
Inputs: @message, @username, @friend_username @adder_username, @blocked_username, @unfriended_username, @unblocked_username, @message_text, @date, @message_id, @game_name

Process: Players can see their friends in friends pop-up page. In addition, from this page, they can see the status of their friends and message their friends through the pop-up screen on this page. Additionally, players can block, delete and show the profile of their friends by using this page.

SQL Statements:
The user adds a friend
```
INSERT INTO Friended VALUES(@username, @adder_username)
```

User unfriends a friend
```
DELETE FROM Friended
WHERE added_id = @username, adder_id = @unfriended_username
```

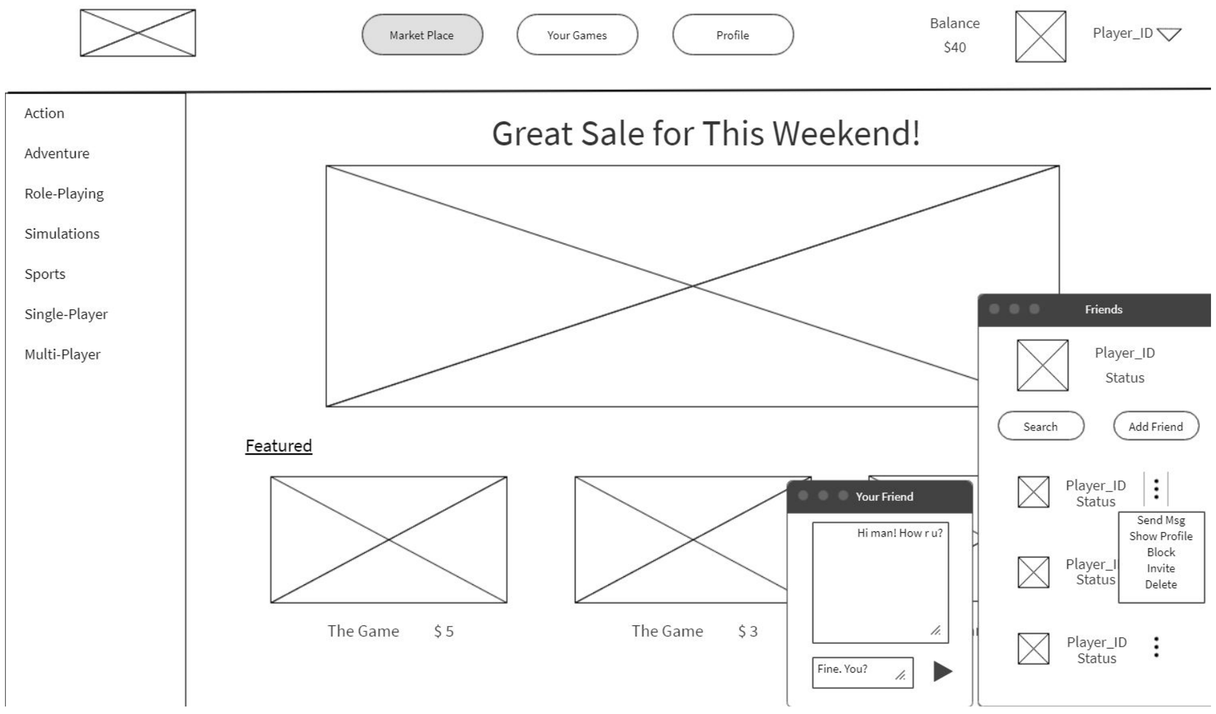
Displaying user's friend list
```
SELECT added_id
FROM Player natural join Friended F
WHERE @username = F.adder_id
```

User blocks a user
```
INSERT INTO Blocked VALUES(@username, @blocked_username)
```

User unblocks a user
```
DELETE FROM Blocked
WHERE blocker_id = @username, blocked_id = @unblocked_username
```

The user writes a message to a friend
```
INSERT INTO Message VALUES(@message_id, @date, @message_text)
INSERT INTO Chats VALUES(@username, @friend_username, @message_id)
```

A premium player can invite another premium_player to a multiplayer game
```
INSERT INTO Multiplayed VALUES(@username, @friend_username, @game_name)
```


Market Place
Your Games
Profile
Balance
$40
Player_ID
Action
Adventure
Role-Playing
Simulations
Sports
Single-Player
Multi-Player
Great Sale for This Weekend!
Featured
The Game $ 5
The Game $ 3
Your Friend
Hi man! How r u?
Fine. You?
Friends
Player_ID
Status
Search
Add Friend
Player_ID
Status
Send Msg
Show Profile
Block
Invite
Delete
Player_I
Status
Player_ID
Status

5.8 Player Account Settings

Inputs: @player_email, @password, @new_email, @new_password

Process: When users open account settings page they can change their email, passwords. They can also see their status(premium or normal) and a number of warnings that is done by the admin. The database will not be modified until player clicks the save changes button.

SQL Statements:

User changes their email

```
UPDATE Information
SET player_email = @new_email
WHERE player_id = @username
```

User changes their password

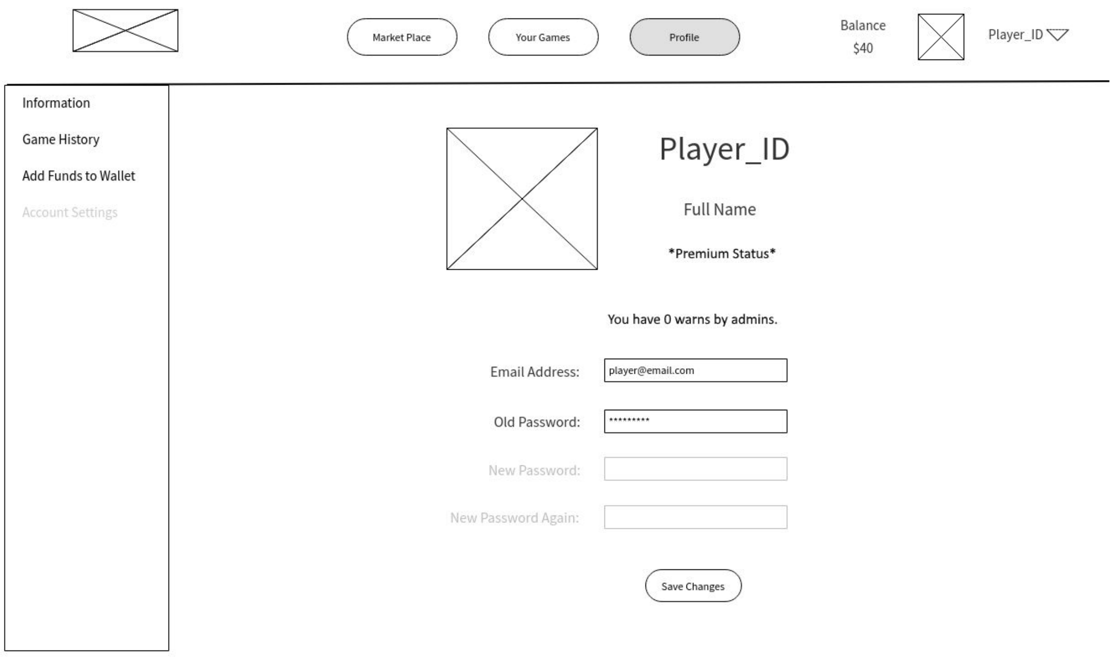
```
UPDATE Player
SET password = @new_password
WHERE player_id = @username
```

The user checks their premium status

```
SELECT premium_status
FROM Player
WHERE player_id = @username
```

The user checks their warning count

```
SELECT count( count)
FROM Player natural join Warned
WHERE player_id = @username
```



Player_ID

Full Name

Premium Status

You have 0 warns by admins.

Email Address:	player@email.com
Old Password:	*********
New Password:	
New Password Again:	

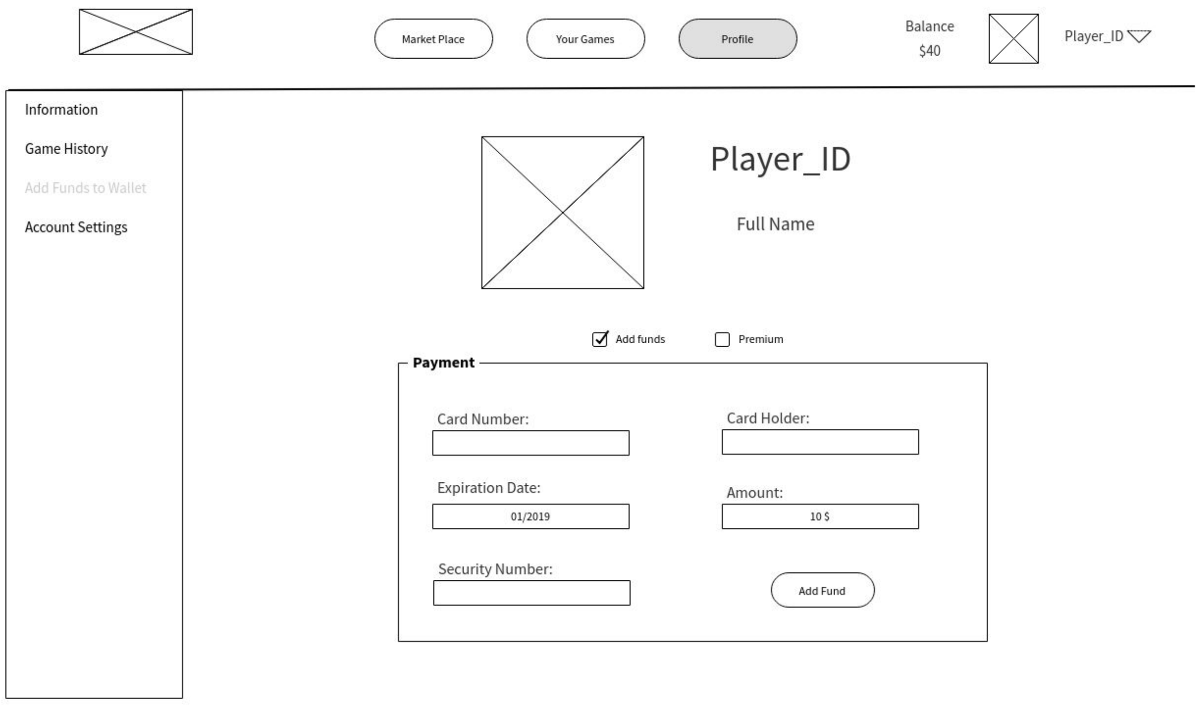
5.9 Add Fund

Inputs: @amount, @username

Process: Player can increase their balance by filling the credit card information on this page.

SQL Statements:

```
UPDATE Player
SET balance = (balance + @amount)
WHERE player_id = @username
```



Expiration Date: 01/2019

Amount: 10 $

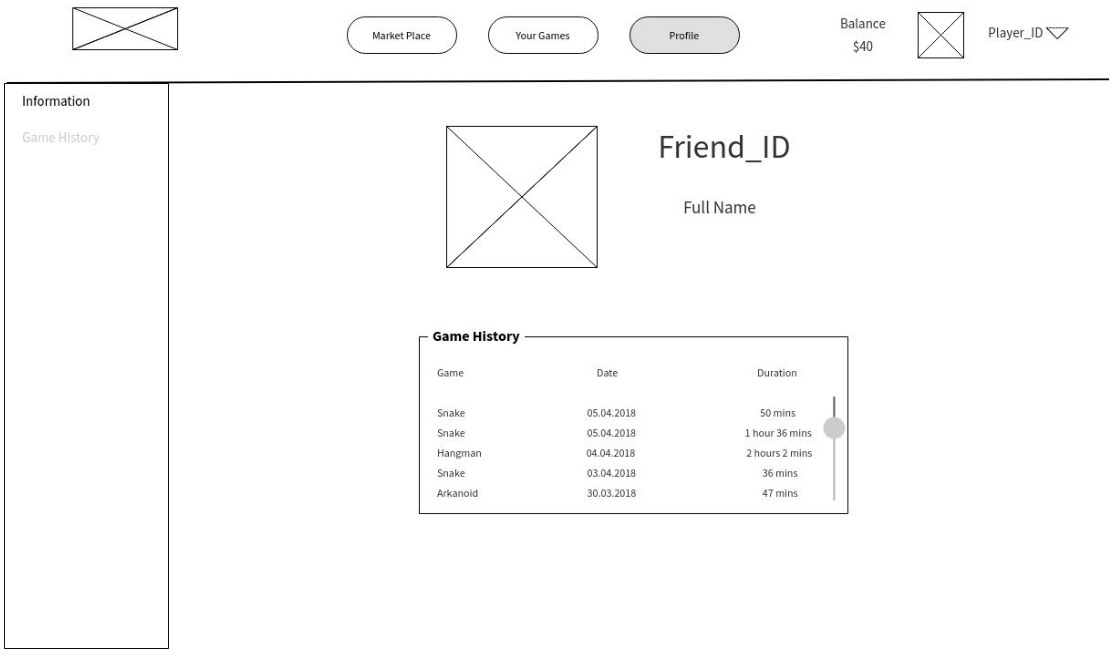
5.10 Friend Profile History

Inputs: @friend_username

Process: Each player can look at their friends' game histories. Game history page includes the latest played games, the date, and duration of the play.

SQL Statements:

```
SELECT game_name, played_date, duration
FROM Played
WHERE player_id = @friend_username
```



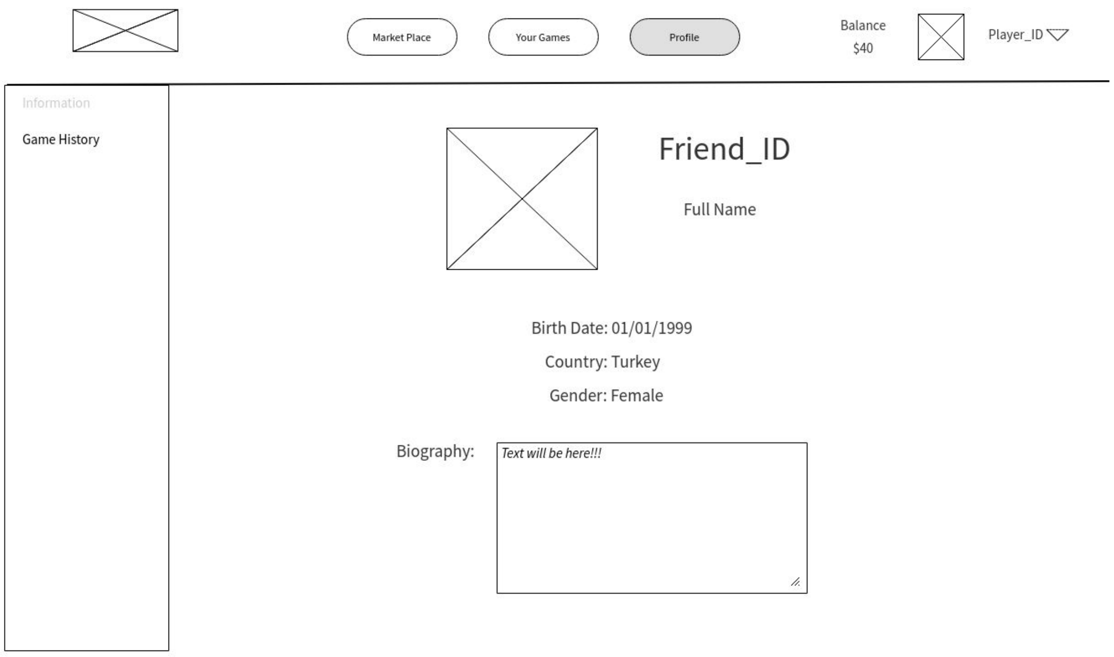
Game	Date	Duration
Snake	05.04.2018	50 mins
Snake	05.04.2018	1 hour 36 mins
Hangman	04.04.2018	2 hours 2 mins
Snake	03.04.2018	36 mins
Arkanoid	30.03.2018	47 mins

5.11 Friend Profile Information
Inputs: @username, @friend_username

Process: Players can see their friends' information by looking their profile. Friend Profile Page includes birth date, country, gender and biography of the friend.

SQL Statements:

```
SELECT player_id, player_email, full_name, birth_date, gender, country, biography
FROM Friended F natural join Information I
WHERE F.added_id = @username, I.player_id = @friend_username
```



5.12 Games
Inputs: @chosen_game_name, @username

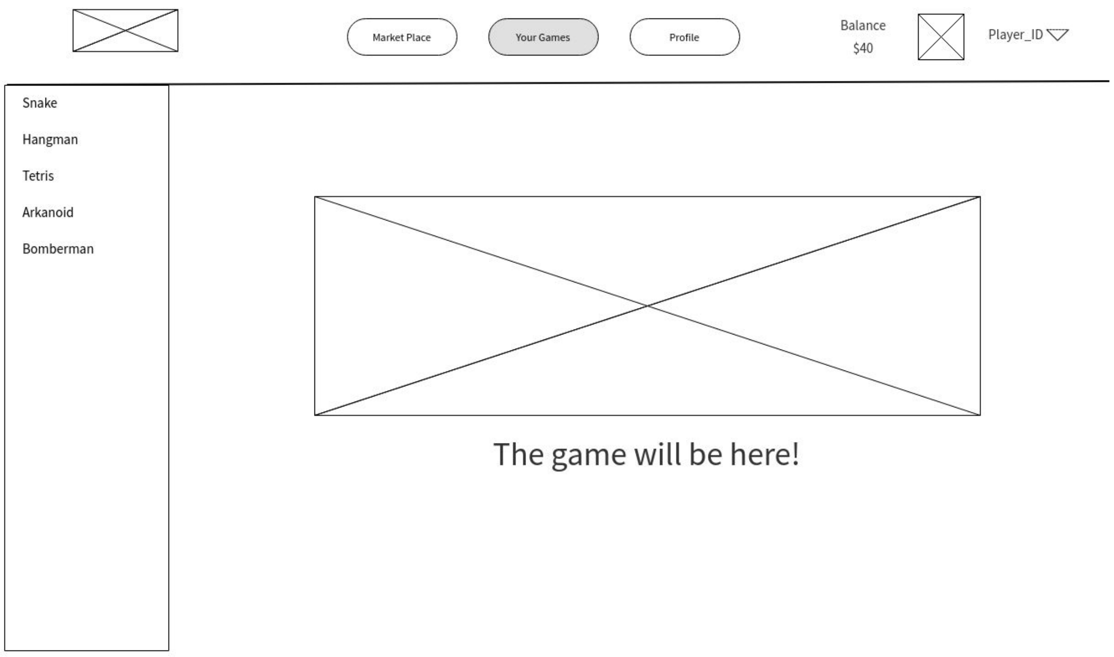
Process: By using “Your Games” button, players can reach the games they bought. From the table at the upper left corner, later players can choose the game they want to play.

SQL Statements:

Player checks the bought game list
SELECT game_name
FROM Bought
WHERE player_id = @username

CREATE VIEW Plays_Game
AS SELECT game_name
FROM BOUGHT
WHERE player_id = @username

Player chooses a game to play
SELECT game_name
FROM Plays_Game
WHERE game_name = @chosen_game_name



5.13 Player Profile History
Inputs: @username

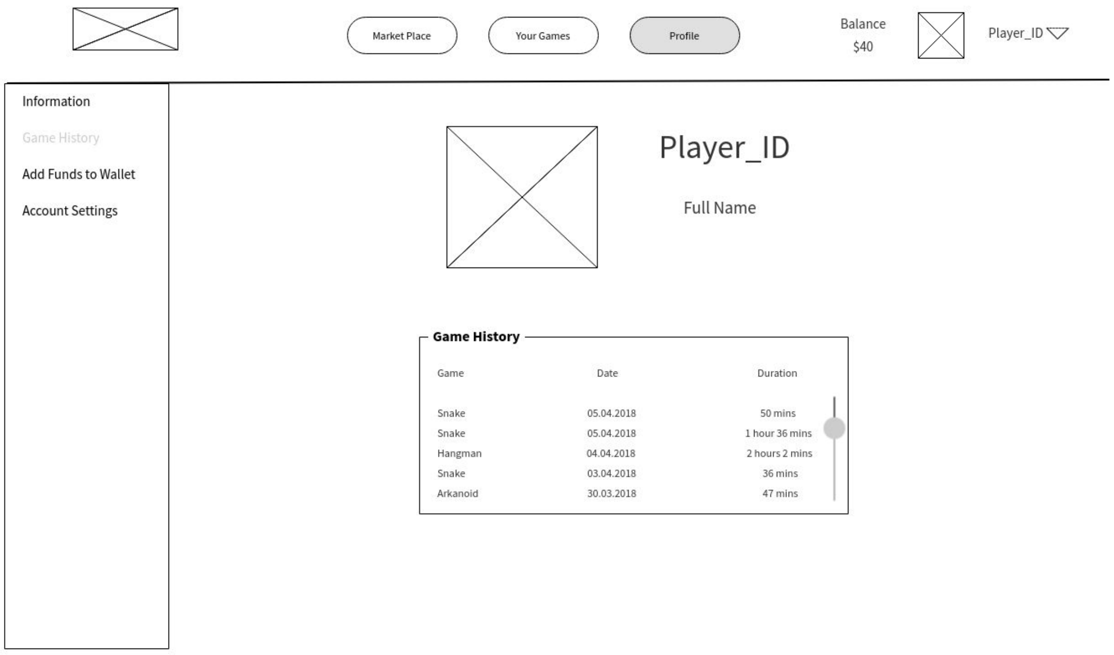
Process: Each player can look at their game history. Game history page includes the latest played games, the date, and duration of the play. In addition, from the table at the left players can go to their information page, funds page and account setting page.

SQL Statements:

The user looks at their game history
```
SELECT game_name, played_date, duration
FROM Played
WHERE player_id = @username
```

The user checks their account settings
```
SELECT player_email, password, premium_status
FROM Player natural join Information
```

User updates premium status
```
UPDATE Player
SET premium_status =  WHEN premium_status = FALSE
          THEN TRUE
END
```



Game History

Game	Date	Duration
Snake	05.04.2018	50 mins
Snake	05.04.2018	1 hour 36 mins
Hangman	04.04.2018	2 hours 2 mins
Snake	03.04.2018	36 mins
Arkanoid	30.03.2018	47 mins

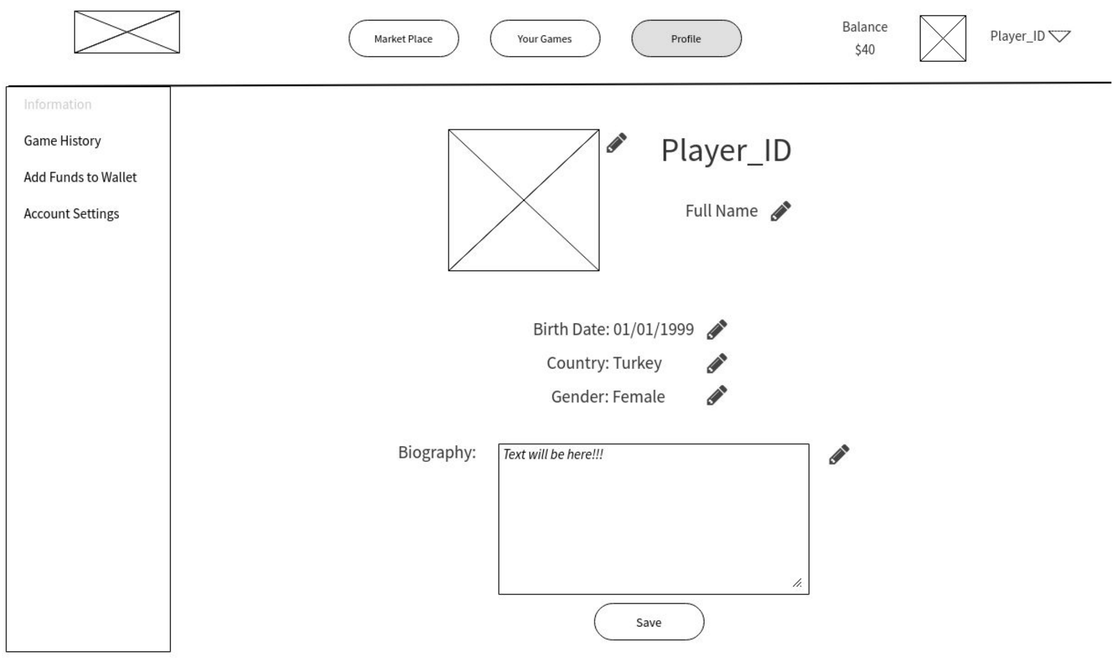
5.14 Player Profile Information
Inputs: @new_biography, @new_country, @new_gender, @new_birth_date, @new_full_name, @new_player_email, @username

Process: Players can see and change their information from their profile page. Profile Page includes birth date, country, gender and biography of the player.

SQL Statements:

User checks their information section
```
SELECT player_id, player_email, full_name, birth_date, gender, country, biography
FROM Information
WHERE player_id = @username
```

User updates information section
```
UPDATE Information
SET player_email = @new_player_email
WHERE player_id = @username

UPDATE Information
SET full_name = @new_full_name
WHERE player_id = @username

UPDATE Information
SET birth_date = @new_birth_date
WHERE player_id = @username

UPDATE Information
SET gender = @new_gender
WHERE player_id = @username

UPDATE Information
SET country = @new_country
WHERE player_id = @username

UPDATE Information
SET biography = @new_biography
WHERE player_id = @username
```



Player_ID

Full Name

Birth Date: 01/01/1999

Country: Turkey

Gender: Female

Biography: *Text will be here!!!*

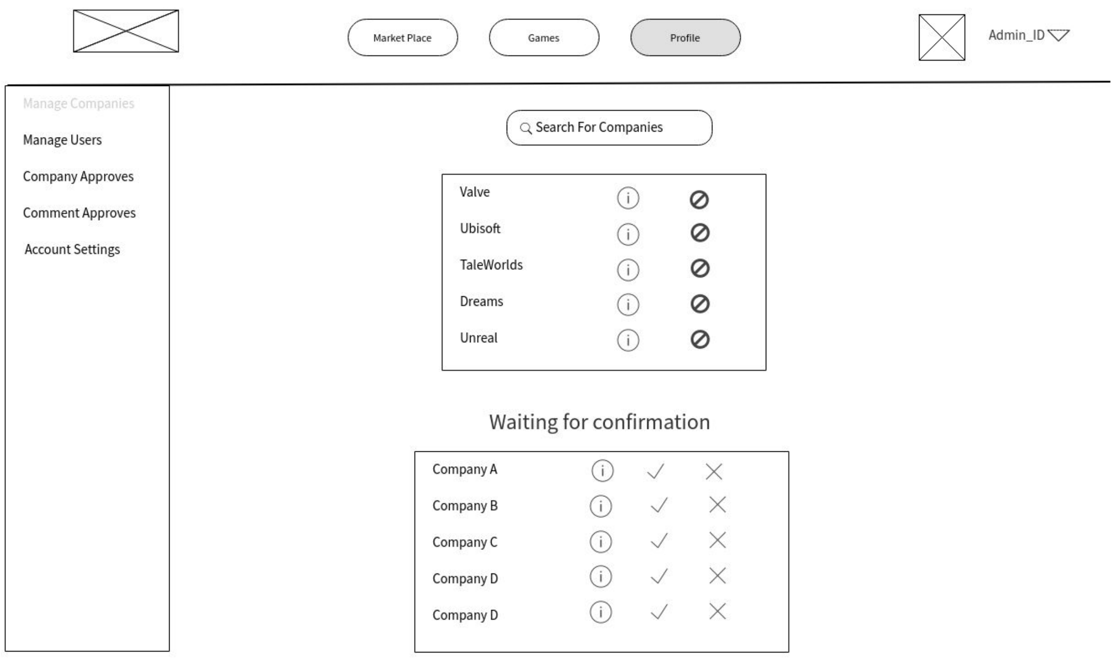
5.15 Profile Manage Companies
Inputs: @company_id, @company_name, @password, @approval_status

Process: From profile manage companies page admins can confirm the new companies and ban the existing companies.

SQL Statements:

Admin confirms the new company
INSERT INTO Company VALUES(@company_id, @company_name, @password, @approval_status)

Admin bans the new company
UPDATE Company
SET approval_status = @approval_status
WHERE company_id = @company_id



5.16 Profile Manage Users

Inputs: @username, @admin_id

Process: From profile manage users page admins can warn and ban the players. The player which is warned several times will be banned by admin.

SQL Statements:

Admin bans a player

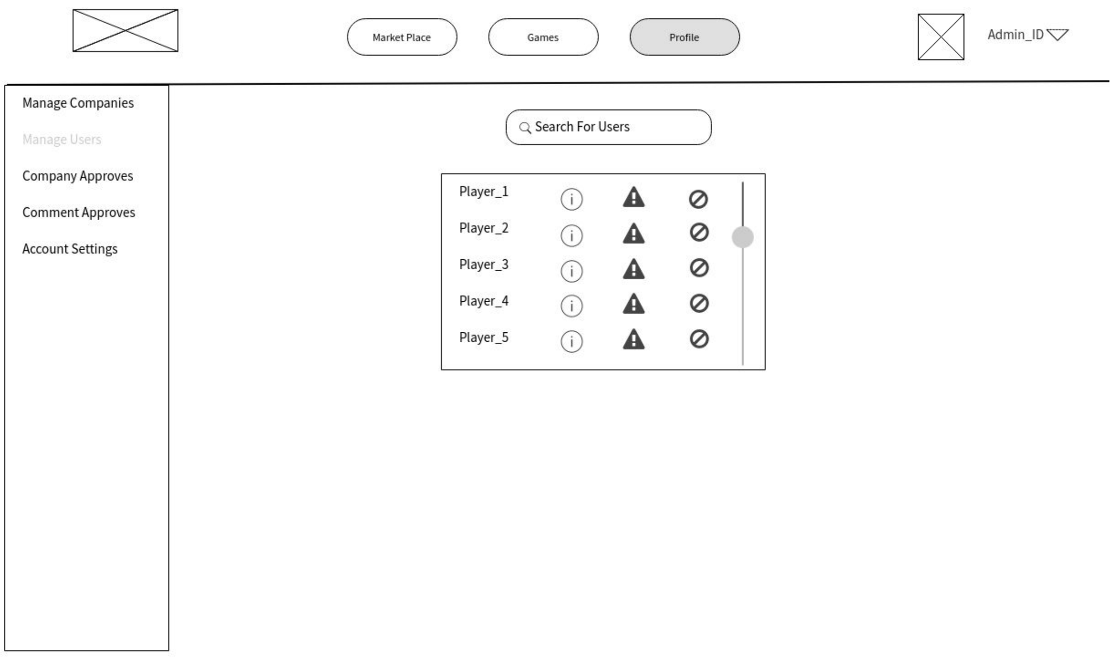
```
INSERT INTO Banned VALUES( @username, @admind_id)
```

Admin warns a player

```
INSERT INTO Warned VALUES( @username, @admind_id, 1)
```

Players warned more than 3 times are banned from the platform

```
CREATE VIEW Count_Table AS
SELECT player_id, count( player_id) as count_number
FROM Player natural join Warned
WHERE player_id = @username


DELETE FROM Player
WHERE player_id = (SELECT CT.player_id
                   FROM Count_Table
                   WHERE count_number = 3)
```



5.17 Profile Manage Comments
Inputs: @comment_id, @comment_text, @comment_date, @like_count, @dislike_count, @approval_status

Process: From own profile page, admins can see the comments made by players to the corresponding games. Admins can confirm the new comments and delete the comments which have unethical elements from this page.

SQL Statements:
Admin sees the list of players, comments and according to game names
SELECT *
FROM Written

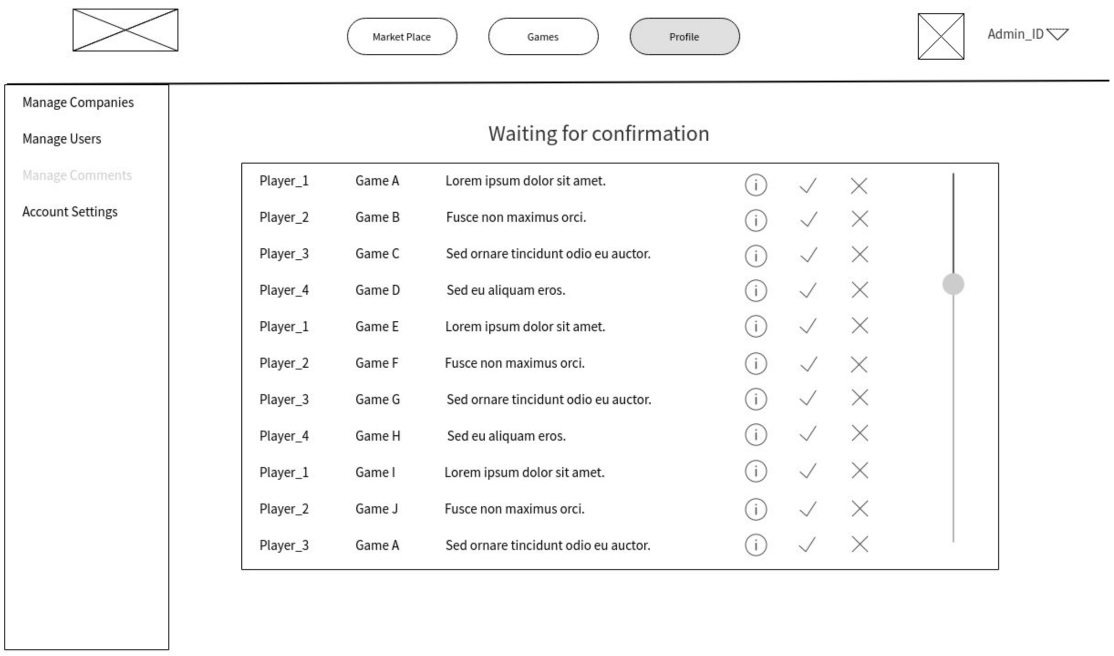
Admin confirms the comment
INSERT INTO Comment VALUES(@comment_id, @comment_date, null, null, TRUE)

Admin does not confirm the comment
INSERT INTO Comment VALUES(@comment_id, @comment_date, null, null, FALSE)

Note: this is what would happen, however, we do not store not confirmed comments in the database

Admin deletes a comment
DELETE FROM Comment

WHERE comment_id = @comment_id



Waiting for confirmation

Player_1	Game A	Lorem ipsum dolor sit amet.	ⓘ	✓	✕
Player_2	Game B	Fusce non maximus orci.	ⓘ	✓	✕
Player_3	Game C	Sed ornare tincidunt odio eu auctor.	ⓘ	✓	✕
Player_4	Game D	Sed eu aliquam eros.	ⓘ	✓	✕
Player_1	Game E	Lorem ipsum dolor sit amet.	ⓘ	✓	✕
Player_2	Game F	Fusce non maximus orci.	ⓘ	✓	✕
Player_3	Game G	Sed ornare tincidunt odio eu auctor.	ⓘ	✓	✕
Player_4	Game H	Sed eu aliquam eros.	ⓘ	✓	✕
Player_1	Game I	Lorem ipsum dolor sit amet.	ⓘ	✓	✕
Player_2	Game J	Fusce non maximus orci.	ⓘ	✓	✕
Player_3	Game A	Sed ornare tincidunt odio eu auctor.	ⓘ	✓	✕

5.18 Admin Profile Settings

Inputs: @new_email, @new_password, @admin_id

Process: When admin opens account settings page they can change their email and password.

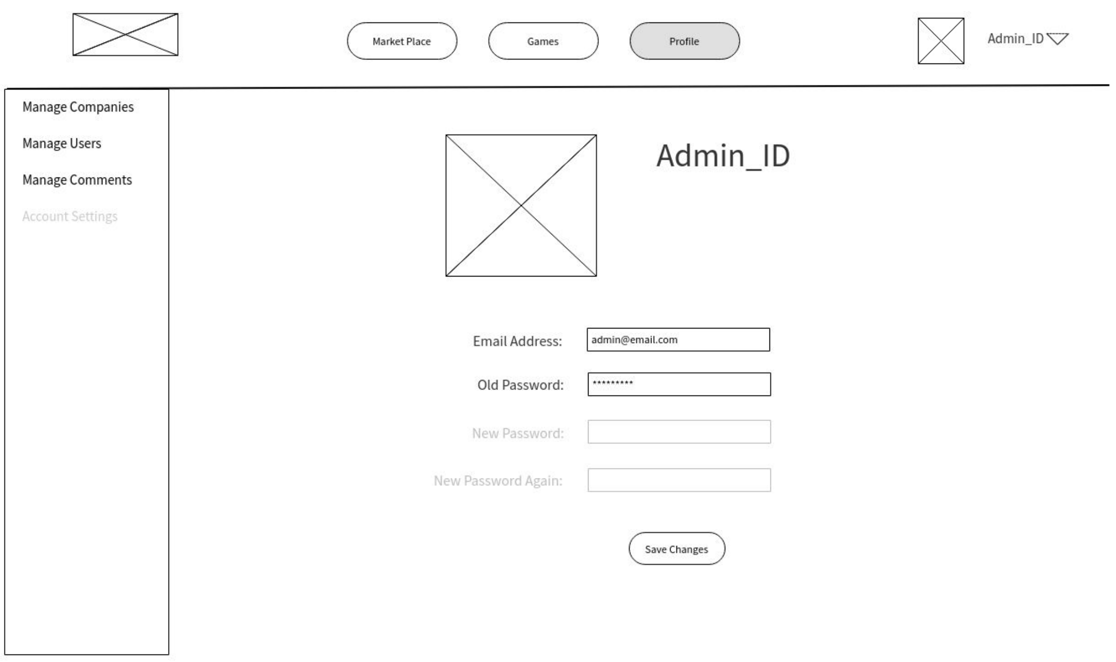
SQL Statements:
Admin sees their email
SELECT admin_email
FROM Admin
WHERE admin_id = @admin_id

Admin sees their password
SELECT password
FROM Admin
WHERE admin_id = @admin_id

Admin changes email
UPDATE Admin

SET admin_email = @new_email
WHERE admin_id = @admin_id

Admin changes password
UPDATE Admin
SET password = @new_password
WHERE admin_id = @admin_id



5.19 Company Event Addition

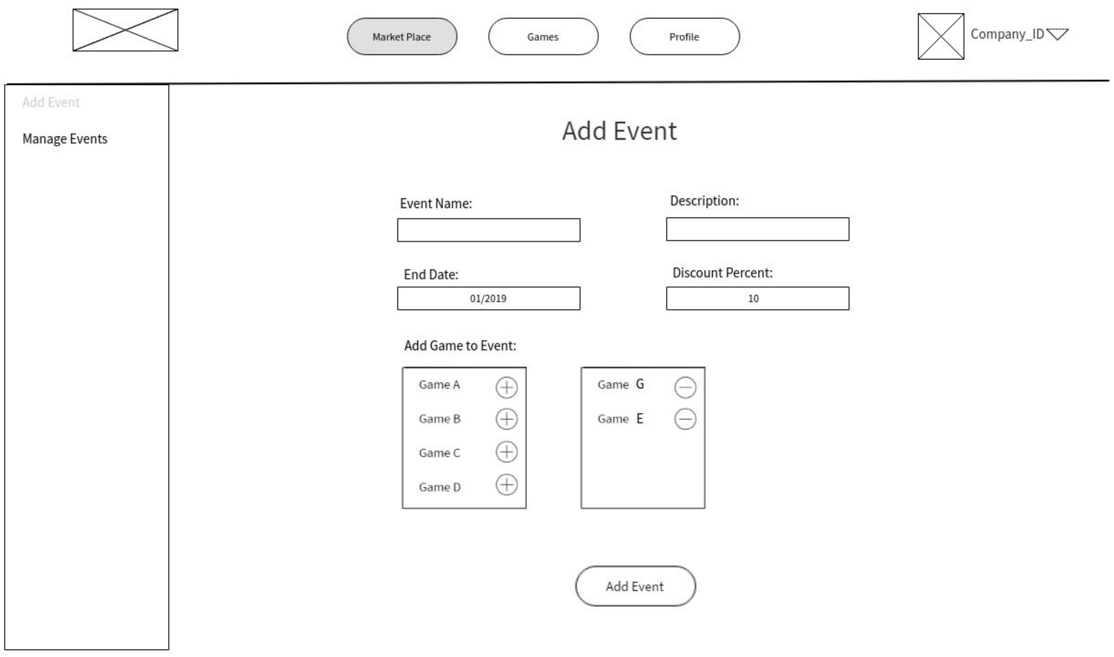
Inputs: @event_id, @event_name, @start_date, @end_date, @description, @game_name, @company_id, @percent

Process: Companies can add a new event and delete an existing event from the marketplace

SQL Statements:
Company adds an event
INSERT INTO Event VALUES(@event_id, @event_name, @start_date, @end_date, @description)

Company adds a discount to the event
INSERT INTO Discount VALUES(@game_name, @company_id, @event_id, @percent)

Company deletes an event
DELETE FROM Event
WHERE event_id = @event_id



5.20 Company Event Management

Inputs: @event_id, @event_name, @start_date, @end_date, @description, @game_name, @company_id, @percent

Process: Companies can manage events by changing their names, dates, and descriptions

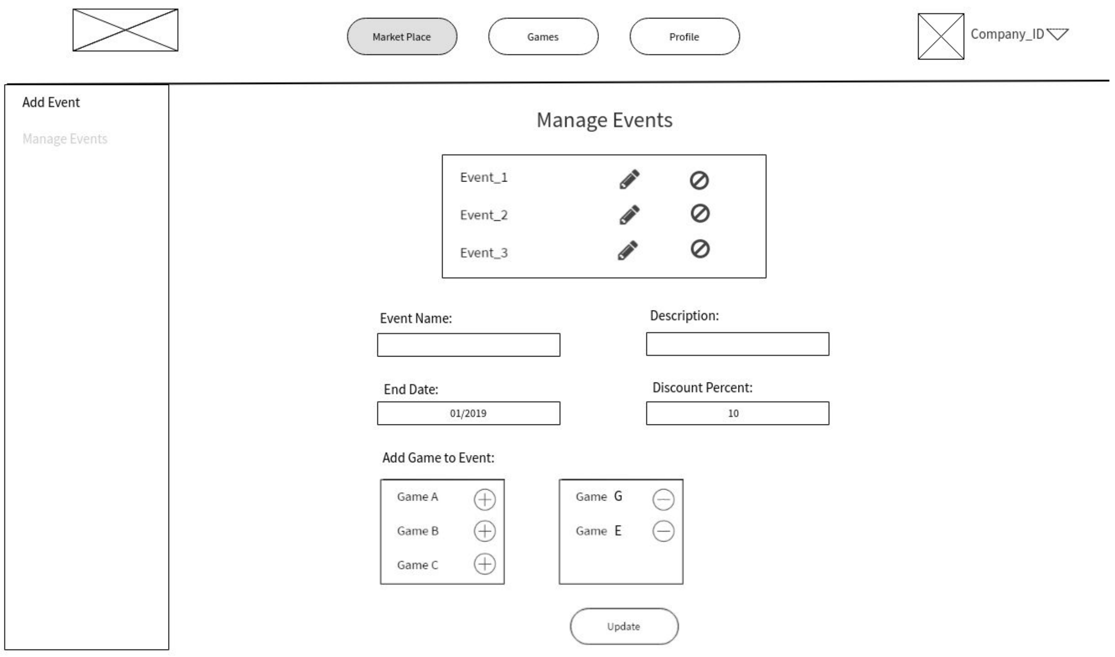
SQL Statements:

Company changes event description
UPDATE Event
SET description = @description
WHERE event_id = @event_id

Company changes event name
UPDATE Event
SET event_name = @event_name
WHERE event_id = @event_id

Company changes event start date
UPDATE Event
SET start_date = @start_date
WHERE event_id = @event_id

Company changes event end date
UPDATE Event
SET end_date = @end_date
WHERE event_id = @event_id



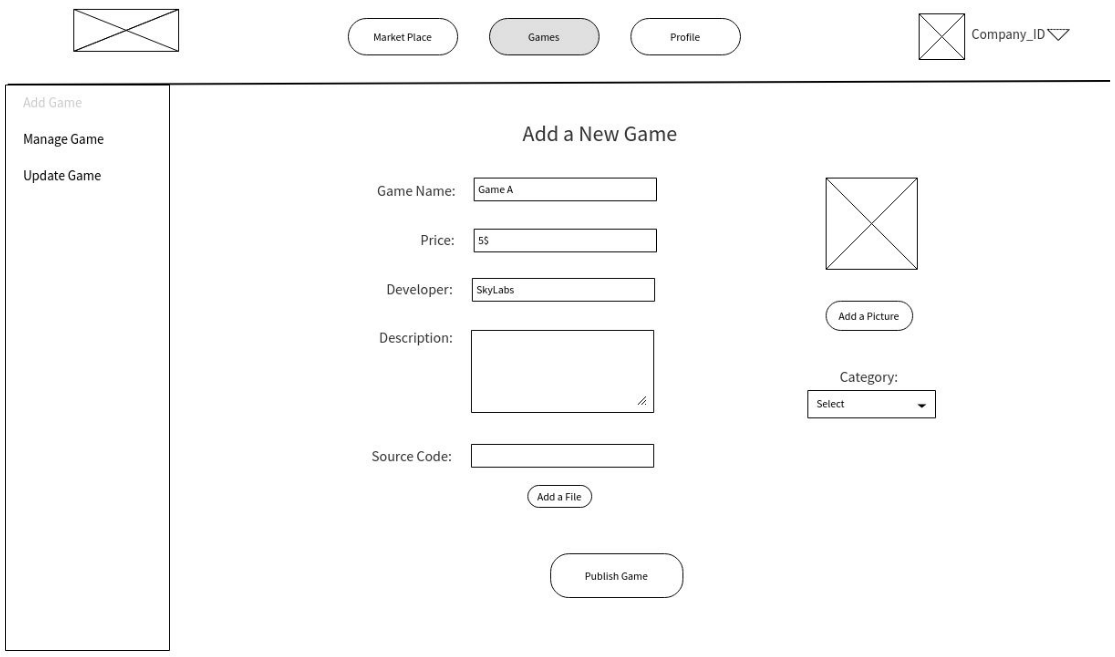
5.21 Company Game Addition

Inputs: @description, @game_name, @price, @developer, @category_name, @picture, @age_limit, @published_date

Process: Companies can add game through add game page by filling the necessary informations.

SQL Statements:
Company adds necessary informations for a new game

INSERT INTO Game VALUES(@game_name, @developer, @description, @published_date, @picture, @price)
INSERT INTO Game_Category VALUES(@category_name, @age_limit)
INSERT INTO Categorized VALUES(@category_name, @game_name)



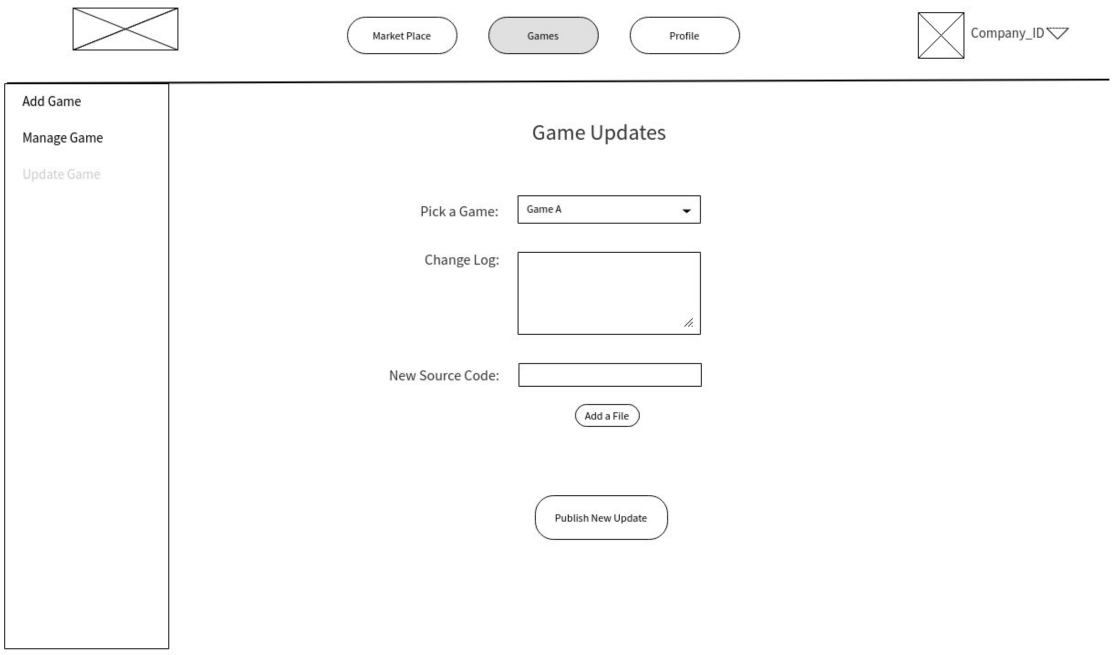
5.22 Company Update Game

Inputs: @game_name, @company_id, @update_date, @change_log, @update

Process: Companies can update their games from this page by inserting new information of the game.
SQL Statements:

Company changes game's update values
INSERT INTO Updates VALUES(@game_name, @company_id, @update_date, @change_log)

Company updates the game
UPDATE Game
SET * = @update
WHERE game_name = @game_name



5.23 Company Manages Game

Inputs: @description, @game_name, @price, @category_name, @picture, @published_date

Process: Companies can manage game through changing informations of the game.

SQL Statements:

Company changes game's description
UPDATE Game
SET description = @description
WHERE game_name = @game_name

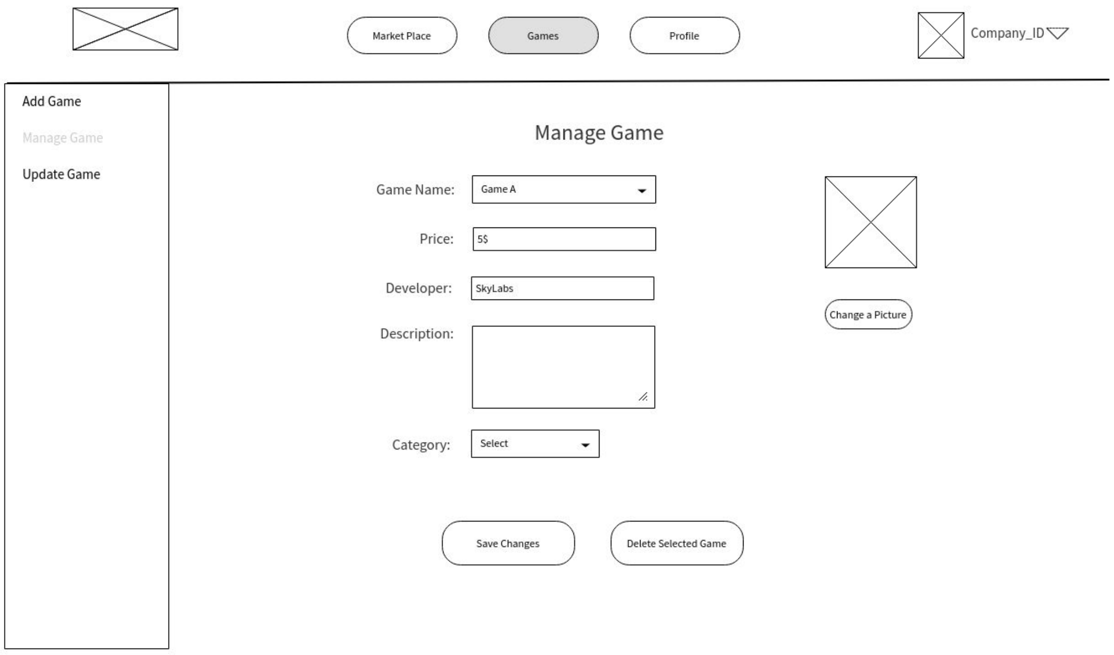
Company changes game's price
UPDATE Game
SET price = @price
WHERE game_name = @game_name

Company changes game's picture

UPDATE Game
SET picture = @picture
WHERE game_name = @game_name

Company changes game's category
UPDATE Categorized
SET category_name = @category_name
WHERE game_name = @game_name

Company changes game's published_date
UPDATE Game
SET published_date = @published_date
WHERE game_name = @game_name



5.24 Company Account Settings
Inputs: @company_email, @password, @new_email, @new_password

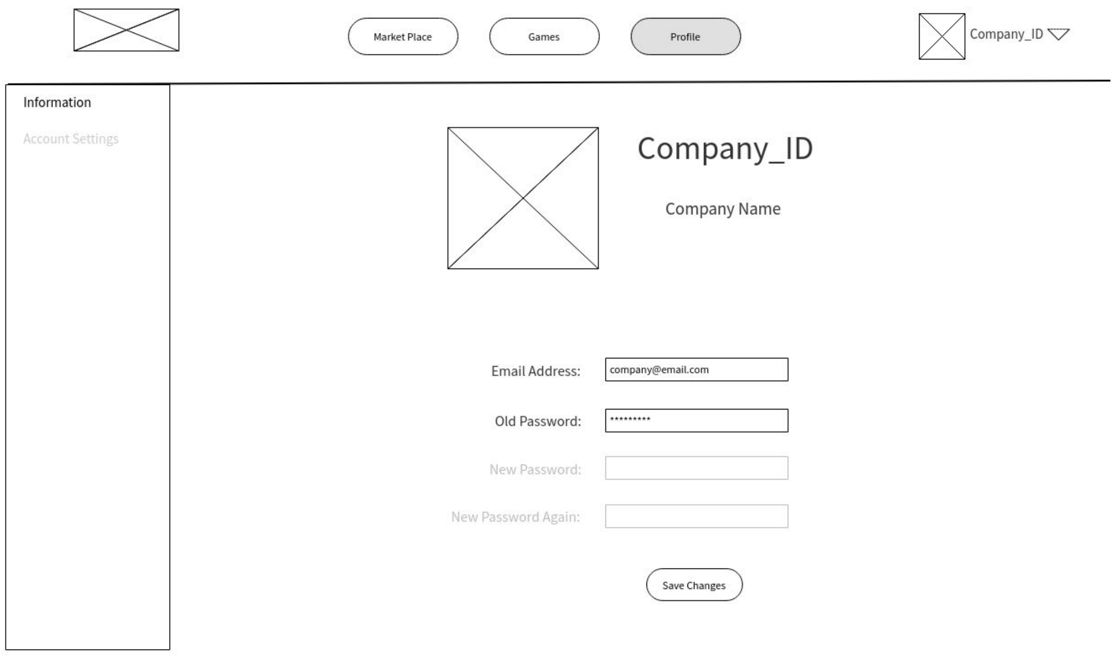
Process: When companies open account settings page they can change their email, passwords. The database will not be modified until player clicks the save changes button.

SQL Statements:

User changes their email
UPDATE Company_Info

SET company_email = @new_email
WHERE company_id = @username

User changes their password
UPDATE Company
SET password = @new_password
WHERE company_id = @username



5.25 Company Profile Information
Inputs: @new_company_webpage,@new_zip, @new_description ,@new_state, @new_city, @new_distinct, @new_picture, @new_full_name, @new_company_email, @new_company_password, @company_id

Process: Companies can see and change their information from their profile page. Profile Page includes company webpage, zip, description, state, city, distinct, picture and name of the company.

SQL Statements:

Company checks their information section

```
SELECT  company_webpage, zip, description, state, city, distinct, picture, full_name
FROM Company_Info
WHERE company_id = @company_id
```

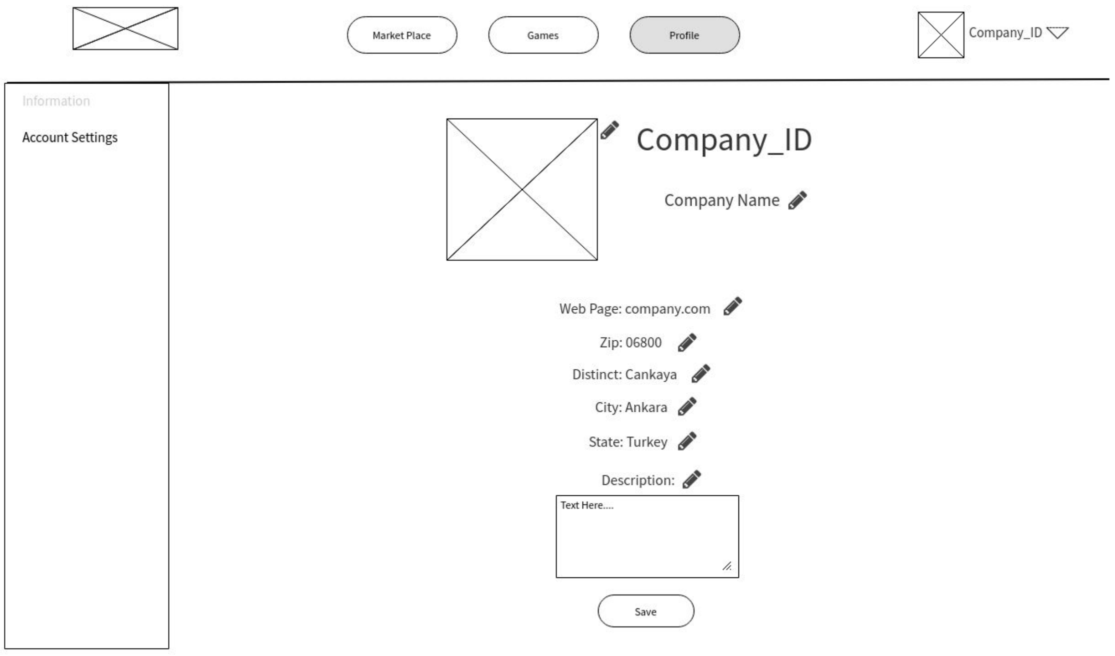
Company updates information section

```
UPDATE Company_Info
SET company_webpage = @new_company_webpage
WHERE company_id = @company_id

UPDATE Company_Info
SET full_name = @new_full_name
WHERE company_id = @company_id

UPDATE Company_Info
SET zip = @new_zip
WHERE company_id = @company_id

UPDATE Company_Info
SET picture = @new_picture
WHERE company_id = @company_id

UPDATE Company_Info
SET description = @new_description
WHERE company_id = @company_id

UPDATE Company_Info
SET state = @new_state
WHERE company_id = @company_id

UPDATE Company_Info
SET city = @new_city
WHERE company_id = @company_id

UPDATE Company_Info
SET distinct = @new_distinct
WHERE company_id = @company_id
```

Company changes password and email

```
UPDATE Company
SET company_email = @new_company_email
WHERE company_id = @company_id
```

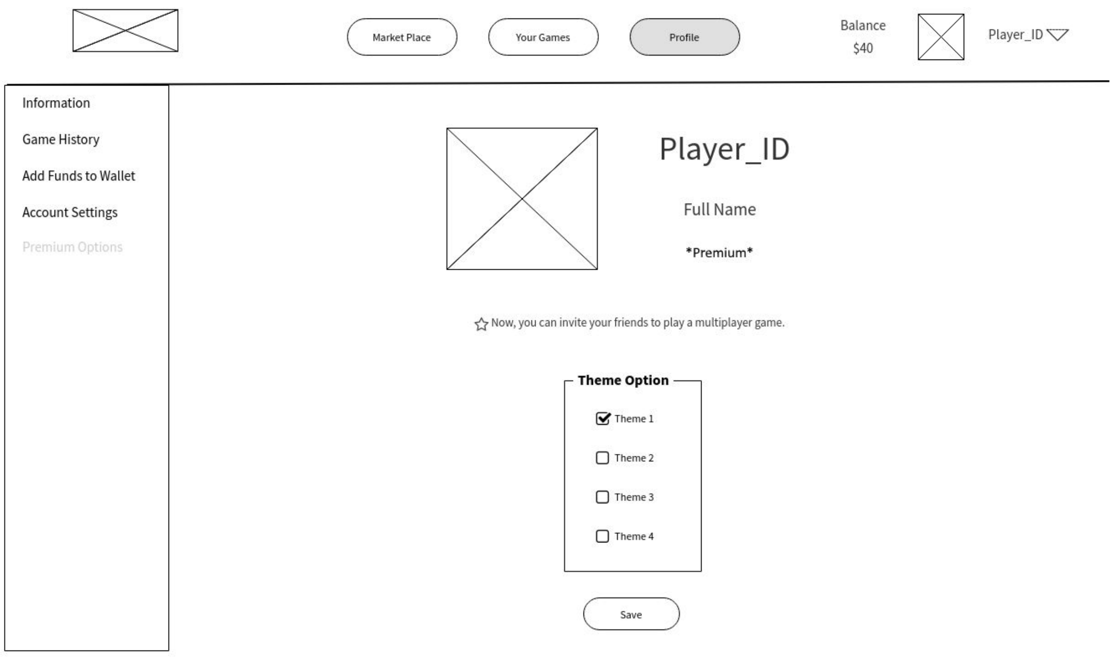
UPDATE Company
SET password = @new_company_password
WHERE company_id = @company_id



5.26 Premium Player Profile
Inputs: @new_theme_option, @username

Process: Premium players can update their background themes

SQL Statements:
UPDATE Premium_Player
SET theme_option = @new_theme_option
WHERE player_id = @username



5.27 Player Writes Comment
Inputs: @comment_id, @comment_text, @comment_date, @like_count, @dislike_count, @approval_status, @username, @game_name

Process: Players can add a comment under a game and wait for it to be approved. They can also see the comment list for a game.

SQL Statements:
Player sees a comment list of a game
SELECT player_id, comment_text
FROM Written natural join Comment
WHERE game_name = @game_name

Players writes a comment under a game
INSERT INTO Comment VALUES(@comment_id, @comment_date, null, null, FALSE)

NOTE: It is FALSE because the admin should first confirm it.

INSERT INTO Written VALUES(@game_name, @username, @comment_id)

6. Advanced Database Components

6.1 Views

6.1.1 Plays_Game

Plays_Game view is created for players to see the game list and choose the game from the table of already bought games easier

```
CREATE VIEW Plays_Game
AS SELECT game_name
FROM BOUGHT
WHERE player_id = @username
```

6.1.2 Count_Table

Count_Table is created for admins to see the list of players who are already warned 3 times and should be banned easier.

```
CREATE VIEW Count_Table AS
SELECT player_id, count( player_id) as count_number
FROM Player natural join Warned
WHERE player_id = @username
```

6.1.3 total_sale_of_company

total_sale_of_company is created for companies to easily get their reports about the total sale of the created event

```
CREATE VIEW total_sale_of_company
AS  ( SELECT D.event_id, count( game_name)
FROM Company C natural join Discount D
GROUP BY D.event_id)
```

6.1.4 game_age_statistics

game_age_statistics is created for companies to easily get their reports about the game and the age statistics

```
CREATE VIEW game_age_statistics
AS (SELECT game_name, (date - birth_date) as age, count( player_id)
FROM Player P natural join Information natural join Played PL
GROUP BY game_name)
```

6.1.5 category_statistics

category_statistics is created for users to see the most popular game categories

```
CREATE VIEW category_statistics AS
(SELECT category_name, count(player_id)
FROM Categorized natural join Played
GROUP BY category_name DESC)
```

6.1.6 player_category_statistics

player_category_statistics is created for players to see their most used categories.

```
CREATE VIEW player_category_statistics AS
(SELECT player_id, count( category_name) as ctgr_name
FROM Categorized natural join Played natural join Player
GROUP BY player_id DESC)
```

6.1.7 most_played_game

most_played_game is created for premium players to see their most invited game list

```
CREATE VIEW most_played_game_statistics AS
(SELECT invited_id, game_name, count( game_name) as game_count
FROM Invited
GROUP BY invited_id, game_name DESC)
```

6.2 Constraints

- Players cannot login the website if they are banned.
- The company which is disapproved by admin is inhibited to login the system.
- The company in the disapproved list cannot sign up with same company name once more.
- Admin cannot ban a banned player once again.
- The price of the game in the event list cannot be larger than its original price.
- Every game should have a company and category.
- Every information has a corresponding player.
- Users (admins, companies, players) cannot see others passwords.
- The descriptions and game lists can be seen by all people, however, to play a game user should buy the game.
- Companies cannot add an additional category for their game, if a company creates a game with a category which is not found in the category choices, that company should contact the admin, if the admin approves the new category, it will be added to the category list.
- Players cannot message or play with a person that she/he is blocked by.
- A player cannot buy a game which has a price larger than Player's balance.
- A player cannot like/dislike comments more than one time.
- A player cannot buy a game which exceeds her/his age limit.
- Game's published date and message date cannot be postdated.

6.3 Triggers

- When a game is updated, update date should be changed.
- When Likes relation is updated, like_count or dislike_count should be updated.
- When a company creates an event for their games the discount percentages should correspond to the games' unit price.
- When a player adds a fund to the balance, the balance should increase by fund amount.
- When a player buys a game the balance of the player should decrease by the game price amount.

6.4 Stored Procedures

- Players will be notified when they get an invitation.
- Players will be notified when any of admins warns them.
- Players will be notified when another player adds them.
- Players will be notified when they receive a message.
- Admins will be notified when a company wants to sign up to the marketplace.
- Admins will be notified when a new comment is made under any game.
- Admins will be notified when a new event is added to the marketplace.

6.5 Reports

6.5.1 Total game sale in an event for a company

Each company can see the number of games sold in the event they made. Let's say that the company searching for the total sale number has an id @company_id. Then the SQL statement would be as following:

```
CREATE VIEW total_sale_of_company
AS  ( SELECT D.event_id, count( game_name)
FROM Company C natural join Discount D
 GROUP BY D.event_id)



SELECT *
FROM total_sale_of_company
WHERE company_id = @company_id
```

6.5.2 Activation record of Players according to age

Each game has a record of statistics where for every age there is a count of players playing it. Let's say that the company searches the statistics of the game with the name @game_name. Then the SQL statement would be as following:

```
CREATE VIEW game_age_statistics
AS (SELECT game_name, (date - birth_date) as age, count( player_id)
FROM Player P natural join Information natural join Played PL
GROUP BY game_name)

SELECT *
FROM game_age_statistics
WHERE game_name = @game_name
```

6.5.3 Popular categories

Users can see the most five popular category choices of the players. This can be displayed on the platform screen.

```
CREATE VIEW category_statistics AS
(SELECT category_name, count(player_id)
FROM Categorized natural join Played
GROUP BY category_name DESC)

SELECT *
FROM category_statistics
```

6.5.4 Player's category choices

Users may see a player's favorite category type. Let's say a player with @player_id wants to see their most played categories. Then the SQL statements will be as following:

```
CREATE VIEW player_category_statistics AS
(SELECT player_id, count( category_name) as ctgr_name
FROM Categorized natural join Played natural join Player
GROUP BY player_id DESC)

SELECT *
FROM player_category_statistics
WHERE player_id = @player_id
```

6.5.5 Premium Player's most invited games

Since one premium player may invite another premium player into a multiplayer game, a premium player would like to see to which game they are invited the most. Let's say that, a premium player with @player_id wants to see the statistics. Then the code would be as following:

```
CREATE VIEW most_played_game_statistics AS
(SELECT invited_id, game_name, count( game_name) as game_count
FROM Invited
GROUP BY invited_id, game_name DESC)

SELECT game_name, game_count
FROM   most_played_game_statistics
WHERE invited_id = @player_id
```

7.Implementation Plan

While creating our website and the logic of the social gaming marketplace we are planning to use PHP, JavaScript, HTML and CSS. For the database part of the project we are planning to use MySQL Server in order to control our data flow.

8.Website

This design report and other activities of the project are available here:

https://github.com/mertosmandy/Ethereal