



Bilkent University

Department of Computer Engineering

CS353 Project Design Report

Video Game Digital Distribution Service

Section 1 / Group 5:

İrem Seven 21704269

Ataberk Gözkaya 21501928

Münevver Uslukılıç 21602297

İlhan Koç 21603429

Instructor:

Özgür Ulusoy

Table of Contents

1 Revised ER Model	4
1.1 Changes Made	4
1.2 ER Diagram	5
2 Table Schemas	6
2.1 Account	6
2.2 Developer	7
2.3 Publisher	8
2.4 User	9
2.5 Curator	10
2.6 Admin	11
2.7 Game	12
2.8 Mod	13
2.9 Achievements	14
2.10 User-Comment	15
2.11 Curator-Comment	16
2.12 Level	17
2.13 Preferences	18
2.14 ask	19
2.15 develops	20
2.16 updates	21
2.17 adds	22
2.18 publish	23
2.19 follows	24
2.20 user-rates	25
2.21 friendship	26
2.22 wishlist	27
2.23 returns	28
2.24 curator-rates	29
2.25 has-game	30
2.26 has-achievement	31
2.27 make-mod	32
2.28 has-user-mod	33
2.29 has-game-mod	34
3 Functional Dependencies and Normalization Of Tables	34
4 Functional Components	35
4.1 Use Cases / Scenarios	35
4.1.1 Admin	35
4.1.2 Player	36
4.1.3 Developer	39

4.1.4 Publisher	41
4.1.5 Curator	43
4.2 Algorithms	45
4.2.1 Displaying Game Algorithms	45
4.2.2 Login Requirements	45
4.3 Data Structures	45
5 User Interface Design and Corresponding SQL Statements	46
5.1 Sign In / Login	46
5.2 Sign Up	47
5.3 Selecting Account Type	48
5.4 Main Page for Users	49
5.5 Game View for Users	50
5.6 User Profile Showing Owned Games	51
5.7 Built Mode Page for Users	52
5.8 Downloaded Modes of User	53
5.9 Comment and Rate by a Curator	55
5.10 Developers Ask for Publish	56
5.11 Developers Updating a Game	58
5.12 Publisher Company Viewing Publish Requests	59
5.13 Extra Functionality: Achievements	60
6 Advanced Database Components	61
6.1 Views	61
6.2 Stored Procedures	61
6.3 Triggers	62
6.4 Constraints	62
7 Implementation Plan	62
8 Website	63

1 Revised ER Model

1.1 Changes Made

In order to improve our previous database structure design, we have made some changes to our previous ER model. Below it is given the changes we made to our previous ER model.

- Name of the update relation was changed to updates.
- Name of the gameOfUser, commentedGame and modOfGame relations were changed to has.
- Name of the modOfUser relation was changed to makes.
- Name of status attribute in friend relation changed into f-status.
- Relation named develops was added in between Game and Publisher entities.
- Relation named returns was added in between Game and User entities. It also has two attributes named return-date and r-status.
- Relation named downloads was added in between User and Mod entities.
- Comments entity divided into two different entities which are named as User Comment and Curator Comment.
- Attributes named version and date were added to updates relation.
- Attribute named u-rate was added to the rates relation between User and Game.
- Attribute named c-rate was added to the rates relation between Curator and Game.
- Attribute named buy-date was added to the has relation between User and Game.
- Attribute rate was deleted from the Game entity since we plan to reach that information by making calculations.
- Attribute noOfFriends was deleted from the User entity since we plan to reach that information by making calculations.
- Attribute noOfFollowers was deleted from the Curator entity since we plan to reach that information by making calculations.
- Attributes level and points were deleted from the User entity.
- Attributes named ach-name, ach-description, and ach-reward were added to the Achievement entity.
- Entity named Level was created. It is binded in a ternary relation with User and Achievement entities via relation has. Thus the earned relation was removed.
- New entity named as Admin added and it is bound to Achievement via adds relation.
- Total participation constraint was made between Game entity and publish relation.
- The cardinality constraint of updates relation was made one to many.

1.2 ER Diagram

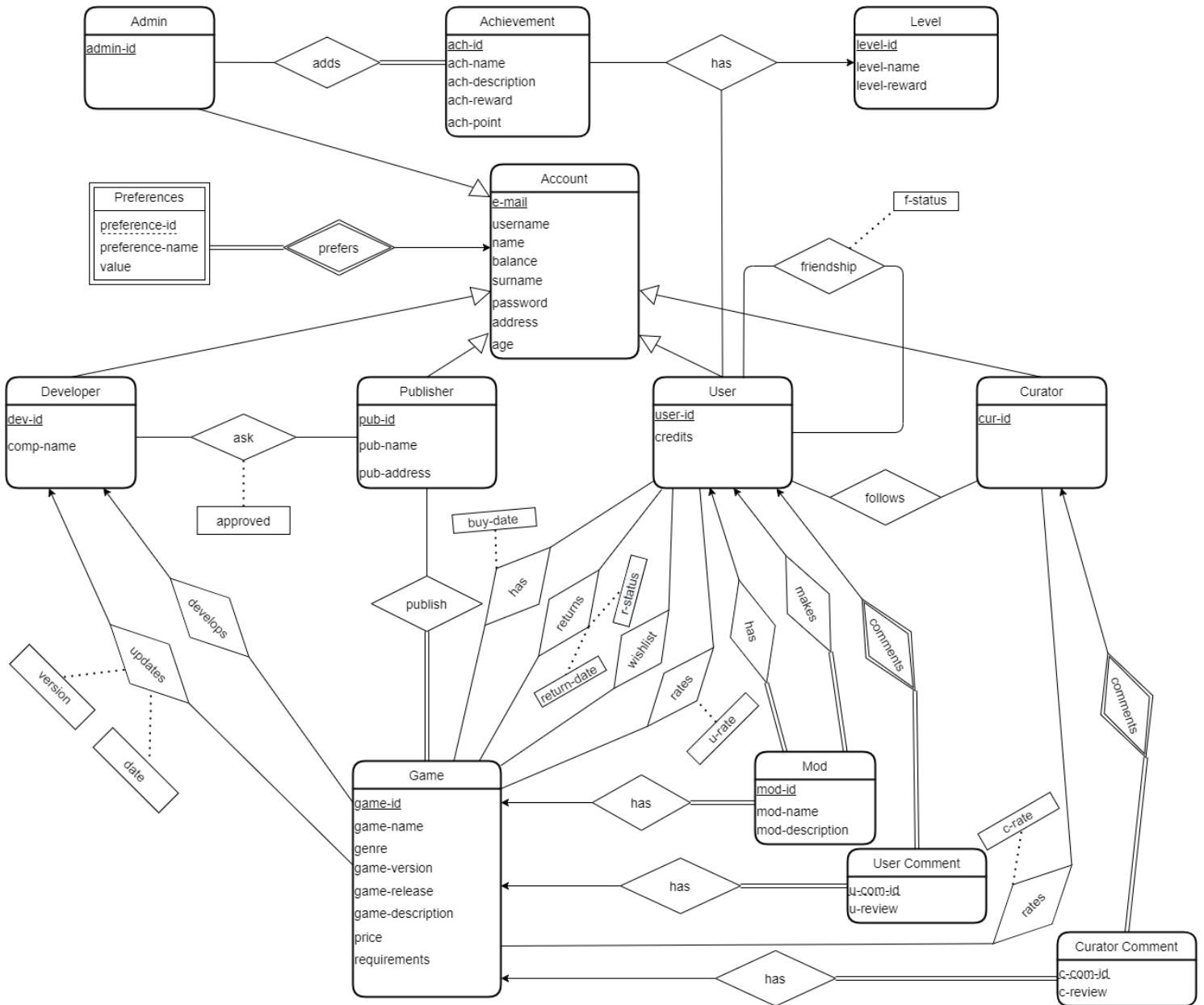


Figure 1 : ER Diagram

2 Table Schemas

2.1 Account

Relational Model

Account(e-mail, username, name, balance, surname, password, address, age)

Functional Dependencies

e-mail → username name balance surname password address age

Candidate Keys

{{e-mail}}

Normal Form

3NF

Table Definition

Create table Account(

e-mail	varchar(30) not null
username	varchar(15) not null
name	varchar(30) not null
balance	int not null
surname	varchar(20) not null
password	varchar(15) not null
address	varchar(80)
age	int
primary key (e-mail)	

);

2.2 Developer

Relational Model

Developer(dev_id, comp-name, e-mail)

FK: e-mail references Account

Functional Dependencies

dev_id \rightarrow comp-name e-mail

Candidate Keys

{{dev_id}}

Normal Form

3NF

Table Definition

```
Create table Developer(  
    dev_id          int not null auto_increment,  
    comp-name       varchar(40) not null,  
    e-mail          varchar(30) not null,  
    foreign key(e-mail) references Account(e-mail),  
    primary key (dev_id)  
);
```

2.3 Publisher

Relational Model

Publisher(pub_id, pub-name, pub-address, e-mail)

FK: e-mail references Account

FK: pub-address references Account(address)

Functional Dependencies

pub_id → pub-name pub-address e-mail

Candidate Keys

{(pub_id)}

Normal Form

3NF

Table Definition

Create table Publisher(

pub_id int not null auto_increment,

pub-name varchar(40) not null,

pub-address varchar(80) not null,

e-mail varchar(30) not null,

foreign key(pub-address) **references** Account(address),

foreign key(e-mail) **references** Account,

primary key (pub_id)

);

2.4 User

Relational Model

User(user_id, credits, e-mail)

FK: e-mail references Account

Functional Dependencies

user_id \rightarrow credits e-mail

Candidate Keys

{{user_id}}

Normal Form

3NF

Table Definition

```
Create table User(  
    user-id          int not null auto_increment,  
    points           int not null,  
    noOfFriends      int not null,  
    level            int not null,  
    e-mail           varchar(30) not null,  
    foreign key(e-mail) references Account(e-mail),  
    primary key (user-id)  
);
```

2.5 Curator

Relational Model

Curator(cur-id, e-mail)

FK: e-mail references Account

Functional Dependencies

cur_id \rightarrow e-mail

Candidate Keys

{{cur-id}}

Normal Form

3NF

Table Definition

```
Create table Curator(  
    cur-id                int not null auto_increment,  
    e-mail                varchar(30) not null,  
    foreign key(e-mail) references Account(e-mail),  
    primary key (cur-id)  
);
```

2.6 Admin

Relational Model

Admin(admin-id, e-mail)

FK: e-mail references Account

Functional Dependencies

admin_id \rightarrow e-mail

Candidate Keys

{{admin-id}}

Normal Form

3NF

Table Definition

```
Create table Admin(  
    admin-id          int not null auto_increment,  
    e-mail            varchar(30) not null,  
    foreign key(e-mail) references Account(e-mail),  
    primary key (admin-id)  
);
```

2.7 Game

Relational Model

Game(game-id, game-name, genre, game-version, game-release, game-description, price, requirements)

Functional Dependencies

game-id \rightarrow game-name, genre, game-version, game-release, game-description, price, requirements

Candidate Keys

{(game-id)}

Normal Form

3NF

Table Definition

```
Create table Game(  
    game-id          int not null auto_increment,  
    game-name        varchar(40) not null,  
    genre            varchar(15),  
    game-version      varchar(10),  
    game-release      date,  
    game-description  varchar(300),  
    price            int,  
    requirements      varchar(100),  
    primary key (game_id)  
);
```

2.8 Mod

Relational Model

Mod(mod-id, mod-name, mod-description, game-id)

FK: game-id references Game

Functional Dependencies

mod-id \rightarrow mod-name, mod-description

Candidate Keys

{(mod-id)}

Normal Form

3NF

Table Definition

Create table Mod(

 mod-id int not null auto_increment,

 game-id int not null,

 mod-name varchar(30) not null,

 mod-description varchar(200) not null,

primary key (mod-id),

foreign key(game-id) **references** Game(game_id)

);

2.9 Achievements

Relational Model

Achievements(ach-id, ach-name, ach-description, ach-reward, ach-point, level-id)

Functional Dependencies

ach-id → ach-name ach-description ach-reward ach-point

Candidate Keys

{{ach-id}}

Normal Form

3NF

Table Definition

```
Create table Achievements(  
    ach-id            int not null auto_increment,  
    ach-name          varchar(40) not null,  
    ach-description    varchar(40),  
    ach-reward         int not null,  
    ach-point          int not null,  
    level-id          int not null,  
    primary key (ach_id),  
    foreign key(level-id) references Level(level-id)  
);
```

2.10 User-Comment

Relational Model

User Comment(user-id, u-com-id, u-review, game-id)

FK: user-id references User

FK: game-id references Game

Functional Dependencies

user-id u-com-id \rightarrow u-review game-id

Candidate Keys

{{user-id, u-com-id}}

Normal Form

3NF

Table Definition

```
Create table User Comment(  
    user-id        int not null,  
    u-com-id       int not null auto_increment,  
    u-review       varchar(250) not null,  
    game-id        int not null,  
    primary key (user-id, u-com-id)  
    foreign key(user-id) references User,  
    foreign key(game-id) references Game  
);
```

2.11 Curator-Comment

Relational Model

Curator Comment(cur-id, c-com-id, c-review, game-id)

FK: cur-id references Curator

FK: game-id references Game

Functional Dependencies

cur-id c-com-id \rightarrow c-review game-id

Candidate Keys

{(cur-id, c-com-id)}

Normal Form

3NF

Table Definition

```
Create table Curator Comment(  
    cur-id          int not null,  
    c-com-id        int not null auto_increment,  
    c-review        varchar(500) not null,  
    game-id         int not null,  
    primary key (cur-id, c-com-id)  
    foreign key(cur-id) references Curator,  
    foreign key(game-id) references Game  
);
```


2.12 Level

Relational Model

Level(level-id, level-name, level-reward)

Functional Dependencies

level-id \rightarrow level-name level-reward

Candidate Keys

{(level-id)}

Normal Form

3NF

Table Definition

```
Create table Level(  
    level-id      int not null auto_increment,  
    level-name    varchar(20) not null,  
    level-reward  int not null,  
    primary key (level-id),  
    check(level-reward > 0)  
);
```

2.13 Preferences

Relational Model

Preferences(e-mail, preference-id, preference-name, value)

FK: e-mail references Account

Functional Dependencies

e-mail preference-id \rightarrow preference-name, value

Candidate Keys

{(e-mail, preference-id)}

Normal Form

3NF

Table Definition

Create table Preferences(

e-mail	varchar(30) not null,
preference-id	int not null auto_increment,
preference-name	varchar(30) not null,
value	varchar(50) not null,
primary key (e-mail, preference-id),	
foreign key (e-mail) references Account	

);

2.14 ask

Relational Model

ask(dev-id, pub-id, approved)

FK: dev-id references Developer

FK: pub-id references Publisher

Functional Dependencies

dev-id preference-id \rightarrow approved

Candidate Keys

{{dev-id, preference-id}}

Normal Form

3NF

Table Definition

```
Create table ask(  
    dev-id          int not null,  
    pub-id          int not null,  
    approved        int not null,  
    primary key (dev-id, preference-id),  
    foreign key(dev-id) references Developer,  
    foreign key(pub-id) references Publisher  
);
```

2.15 develops

Relational Model

develops(game-id, dev-id)

FK: game-id references Game

FK: dev-id references Developer

Functional Dependencies

game-id \rightarrow dev-id

Candidate Keys

{(game-id)}

Normal Form

3NF

Table Definition

```
Create table develops(  
    game-id      int not null,  
    dev-id       int not null,  
    primary key (game-id),  
    foreign key(dev-id) references Developer,  
    foreign key(game-id) references Game  
);
```

2.16 updates

Relational Model

updates(game-id, dev-id, version, date)

FK: game-id references Game

FK: dev-id references Developer

FK: version references Game(game-version)

Functional Dependencies

game-id \rightarrow dev-id version date

Candidate Keys

{(game-id)}

Normal Form

3NF

Table Definition

```
Create table updates(  
    game-id      int not null,  
    dev-id       int not null,  
    version      varchar(10) not null,  
    date         date not null,  
    primary key (game-id),  
    foreign key(dev-id) references Developer,  
    foreign key(game-id) references Game,  
                on delete cascade  
                on update cascade  
);
```

2.17 adds

Relational Model

adds(admin-id, ach-id)

FK: admin-id references Admin

FK: ach-id references Achievement

Functional Dependencies

None

Candidate Keys

{{admin-id, ach-id}}

Normal Form

3NF

Table Definition

```
Create table adds(  
    admin-id      int not null,  
    ach-id        int not null,  
    primary key (admin-id, ach-id),  
    foreign key(admin-id) references Admin,  
    foreign key(ach-id) references Achievement,  
);
```

2.18 publish

Relational Model

publish(pub-id, game-id)

FK: pub-id references Publisher

FK: game-id references Game

Functional Dependencies

None

Candidate Keys

{{pub-id, game-id}}

Normal Form

3NF

Table Definition

```
Create table publish(  
    pub-id          int not null,  
    game-id         int not null,  
    primary key (pub-id, game-id),  
    foreign key(pub-id) references Publisher,  
    foreign key(game-id) references Game,  
);
```

2.19 follows

Relational Model

follows(user-id, cur-id)

FK: user-id references User

FK: cur-id references Curator

Functional Dependencies

None

Candidate Keys

{{user-id, cur-id}}

Normal Form

3NF

Table Definition

```
Create table follows(  
    user-id      int not null,  
    cur-id       int not null,  
    primary key (user-id, cur-id),  
    foreign key(user-id) references User,  
    foreign key(cur-id) references Curator,  
);
```


2.20 user-rates

Relational Model

user-rates(game-id, user-id, u-rate)

FK: game-id references Game

FK: user-id references User

Functional Dependencies

game-id user-id \rightarrow u-rate

Candidate Keys

{(game-id user-id)}

Normal Form

3NF

Table Definition

```
Create table user-rates(  
    game-id      int not null,  
    user-id      int not null,  
    u-rate       int not null,  
    primary key (game-id, user-id),  
    foreign key(user-id) references User,  
    foreign key(game-id) references Game,  
);
```

2.21 friendship

Relational Model

friendship(user-id, friend-id, f-status)

FK: user-id references User

FK: friend-id references User(user-id)

Functional Dependencies

user-id friend-id \rightarrow f-status

Candidate Keys

{(user-id, friend-id)}

Normal Form

3NF

Table Definition

Create table friendship(

 user-id int not null,

 friend-id int not null

 f-status int null,

primary key (user-id, friend-id),

foreign key(user-id) **references** User,

foreign key(friend-id) **references** User(user-id),

);

2.22 wishlist

Relational Model

wishlist(user-id, game-id)

FK: user-id references User

FK: game-id references Game

Functional Dependencies

None

Candidate Keys

{{user-id, game-id}}

Normal Form

3NF

Table Definition

```
Create table wishlist(  
    user-id      int not null,  
    game-id      int not null  
    primary key (user-id, game-id),  
    foreign key(user-id) references User,  
    foreign key(game-id) references Game,  
);
```

2.23 returns

Relational Model

returns(user-id, game-id, return-date, r-status)

FK: user-id references User

FK: game-id references Game

Functional Dependencies

user-id game-id \rightarrow return-date r-status

Candidate Keys

{{user-id, game-id}}

Normal Form

3NF

Table Definition

```
Create table returns(  
    user-id      int not null,  
    game-id      int not null,  
    return-date  date not null,  
    r-status     varchar(20),  
    primary key (user-id, game-id),  
    foreign key(user-id) references User,  
    foreign key(game-id) references Game,  
);
```

2.24 curator-rates

Relational Model

curator-rates(cur-id, game-id, c-rate)

FK: cur-id references Curator

FK: game-id references Game

Functional Dependencies

cur-id game-id \rightarrow return-date c-rate

Candidate Keys

{(cur-id, game-id)}

Normal Form

3NF

Table Definition

```
Create table curator-rates(  
    cur-id          int not null,  
    game-id         int not null,  
    c-rate          int not null,  
    primary key (cur-id, game-id),  
    foreign key(cur-id) references Curator,  
    foreign key(game-id) references Game,  
);
```

2.25 has-game

Relational Model

has-game(user-id, game-id, buy-date)

FK: user-id references User

FK: game-id references Game

Functional Dependencies

user-id game-id \rightarrow buy-date

Candidate Keys

{(user-id, game-id)}

Normal Form

3NF

Table Definition

Create table has-game(

 user-id int not null,

 game-id int not null,

 buy-date date not null,

primary key (user-id, game-id),

foreign key(user-id) **references** User,

foreign key(game-id) **references** Game,

);

2.26 has-achievement

Relational Model

has-achievement(user-id, ach-id, level-id)

FK: user-id references User

FK: ach-id references Achievement

FK: level-id references Level

Functional Dependencies

user-id ach-id \rightarrow level-id

Candidate Keys

{(user-id, ach-id)}

Normal Form

3NF

Table Definition

```
Create table has-achievement(  
    user-id      int not null,  
    ach-id       int not null,  
    level-id     int not null,  
    primary key (user-id, ach-id),  
    foreign key(user-id) references User,  
    foreign key(ach-id) references Achievement,  
    foreign key(level-id) references Level,  
);
```

2.27 make-mod

Relational Model

make-mod(mod-id, user-id)

FK: mod-id references Mod

FK: user-id references User

Functional Dependencies

mod-id \rightarrow user-id

Candidate Keys

{(mod-id)}

Normal Form

3NF

Table Definition

```
Create table make-mod(  
    mod-id      int not null,  
    user-id     int not null,  
    primary key (mod-id),  
    foreign key(user-id) references User,  
    foreign key(mod-id) references Mod  
);
```


2.28 has-user-mod

Relational Model

has-user-mod(mod-id, user-id)

FK: mod-id references Mod

FK: user-id references User

Functional Dependencies

mod-id \rightarrow user-id

Candidate Keys

{(mod-id)}

Normal Form

3NF

Table Definition

```
Create table has-user-mod(  
    mod-id          int not null,  
    user-id         int not null,  
    primary key (mod-id),  
    foreign key(user-id) references User,  
    foreign key(mod-id) references Mod  
);
```

2.29 has-game-mod

Relational Model

has-game-mod(mod-id, game-id)

FK: mod-id references Mod

FK: user-id references User

Functional Dependencies

mod-id \rightarrow game-id

Candidate Keys

{(mod-id)}

Normal Form

3NF

Table Definition

```
Create table has-game-mod(  
    mod-id          int not null,  
    game-id         int not null,  
    primary key (mod-id),  
    foreign key(game-id) references Game,  
    foreign key(mod-id) references Mod  
);
```

3 Functional Dependencies and Normalization Of Tables

The relation Schemas in the design report include every functional dependency and normal forms. Because of the relations being all in third normal form (3NF), no decomposition or normalization is necessary.

4 Functional Components

4.1 Use Cases / Scenarios

4.1.1 Admin

Login: Admins login with their email and password that are given by the software engineers.

Manage Accounts: Admins manage all of the user types' accounts in the system in case of any situation related to security of the system. They can reach all of the information other than the credit card number.

View Game: Admins view the games. They can search and categorize them. When they view a game, they can access the information such as release date, version, system requirements of the game. They also can view the modes of the game, curator and player rates and other comments.

View Player: Admins view players' profiles. Their profile includes the information about the games they have, their friends, their game modes.

View Developer: Admins view developers' profiles and check the uploaded games and updates.

View Curator: Admins view curators' profiles and see their comments.

View Publisher: Admins view publishers' profiles and see their published games.

View Game Results: Admins view game results and achievements of the Players in the game.

Add Achievement: Admins add achievements to the players according to their game results and analysis in the system.

Manage Account: Admins manage their account by changing the emails and passwords.

Change Settings: Admins change website colors according to their personal preferences.

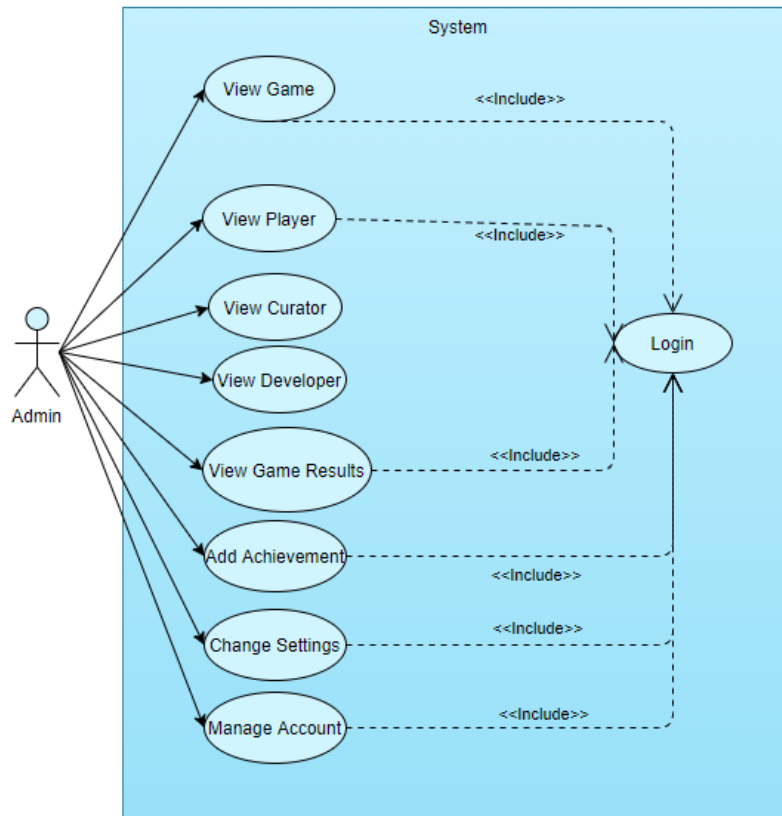


Figure 2 : Admin Use Case Diagram

4.1.2 Player

Sign Up: Users use their email, username, name, surname, age and passwords to create an account on the system. After they choose the Player as their account type, their account is created. In order to purchase, play and rate the games, add friends and do other account based actions on the system, they have to have an account. However, there are a couple of features that they can do without creating an account, such as viewing games, comments of the games and publishers.

Login: Players login to the access to their accounts by using email addresses and the password saved in the database of the system. If they don't have an account, they can sign up and create their account.

View Game: Players view the games. They can search and categorize them. When they view a game, they can access the information such as release date, version, system

requirements of the game. They also can view the modes of the game, curator and player rates and other comments.

Write Comment: Players write comments to the games. Their comments are published on the game page that is associated with the comment.

Rate Game: Players rate the games by giving from 1 to 5 stars to the game and the average is calculated by the stars that all other users of the system have given. This calculation is added to the game pages to help others to review the rate of the games.

Add Game to the Wishlist: Players add games to their wishlist. It is beneficial for viewing the games that they are planning to purchase and check those games' current cost, comments and versions.

Load Money: After logging in and accessing the personal account, players can load money to their account by using their credit cards. They can not have their money back from the account.

Purchase Game: Players purchase games with the money they loaded to their account. If they do not have enough money in their account, the system gives a warning about that. So that they are informed about the purchasing process.

Download Update: Players download the recent update of the games that publishers add to the game pages.

Uninstall Mode: Players uninstall the modes that they downloaded from their library.

Build Mode: Players build mode for existing games in the system. They can choose the mode that they want to play the game before they open their games.

Publish Mode: Players publish the modes that they create and let others have that mode. Their modes are published on the game pages and also on their profile, so that they can find their friends' game modes easily.

Download Mode: Players download the game modes that other players publish.

Return Game: Players return / uninstall the games from their libraries. In the first 2 days their money is returned to their account saved in the system.

Categorize Game: Players categorize games by their maturity contents. So that they create limitations for themselves such as not displaying games that include violence or nudity.

View Player: Players view other players' profiles. Their profile includes the information about the games they have, their friends, their game modes. They also view their own account information by clicking the Profile section on the system.

Add / Remove Friend: Players add friends on the game by clicking to the add / remove friend buttons on other player's profile. Adding friends lets players send messages to each other.

View Publisher: Players view publishers' profiles and view the information about the games that they publish.

View Curators: Players view curators' profiles and view the latest comment that they wrote to the games. By following the curators, they get notified for the comments they publish at that time.

Manage Account: Players manage their account by changing the email, passwords, username, name and surname.

Change Settings: Players change website colors according to their personal preferences.

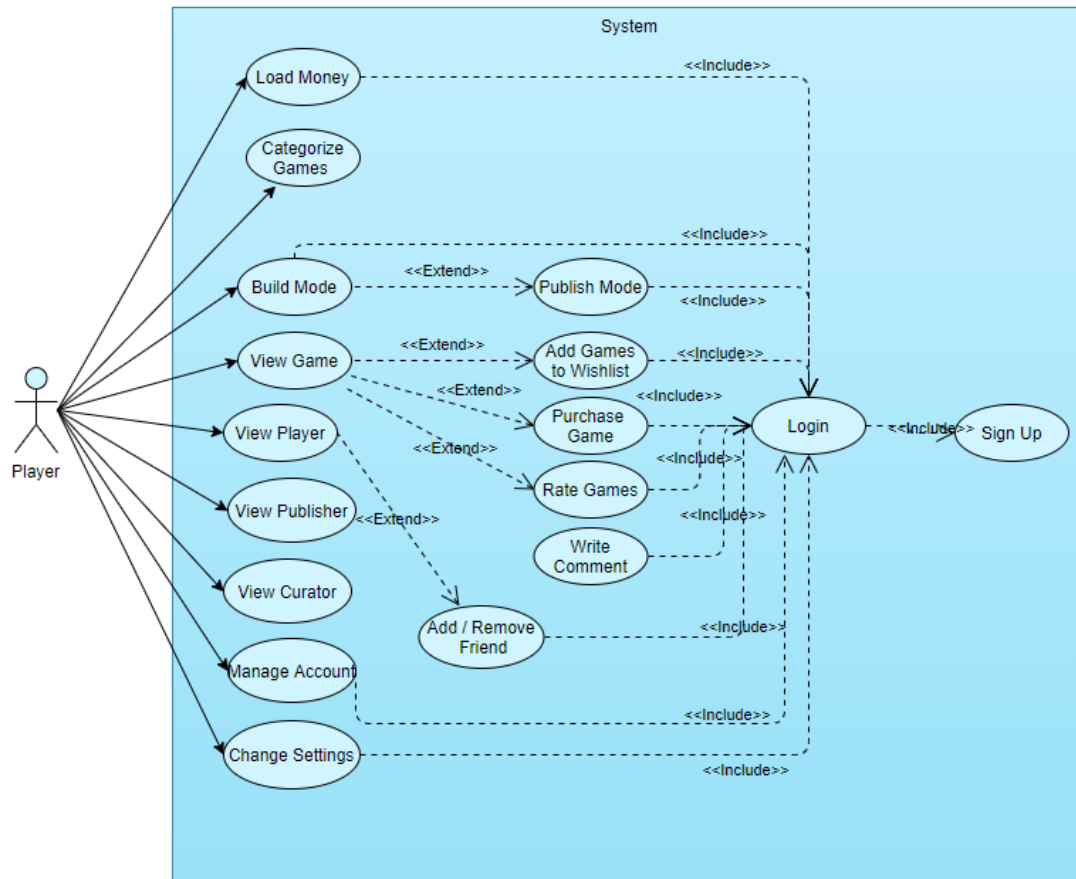


Figure 3 : Player Use Case Diagram

4.1.3 Developer

Sign Up: Developers use their email, username, name, surname, age and passwords to create an account on the system. After they choose the Developer as their account type, their account is created. In order to purchase, play and rate the games, add friends and do other account based actions on the system, they have to have an account. However, there are a couple of features that they can do without creating an account, such as viewing games, comments of the games and publishers.

Login: Developers login to the access to their accounts by using email addresses and the password saved in the database of the system. If they don't have an account, they can sign up and create their account.

View Publisher: Developers view publishers' profiles and view the information about the games that they publish.

View Curator: Developers view curators' profiles and view the latest comment that they wrote to the games.

View Game: Developers view the games. They can search and categorize them. When they view a game, they can access the information such as release date, version, system requirements of the game. They also can view the modes of the game, curator and player rates and other comments. Unlike the Players and Curators in the system, they cannot write comments to the games.

Send Request to Publisher: Developers send requests to the publishers, so that publishers get notified for the games that developers want them to publish or update.

Load Game: Developers load games to the system before they send the request to the publisher. When they upload the game, they have to specify the name, description, game category.

Update the Game: Developers update the game by choosing the game they want to upload and the update of the game is seen as the current version of the game on their platform. However, the previous versions are still saved in the database of the system. Developers have to specify the information about the update by writing a description.

Categorize Game: Developers categorize games by their maturity contents. So that they create limitations for themselves such as not displaying games that include violence or nudity.

Manage Account: Developers manage their account by changing the email, passwords, username, name and surname.

Change Settings: Developers change website colors according to their personal preferences.

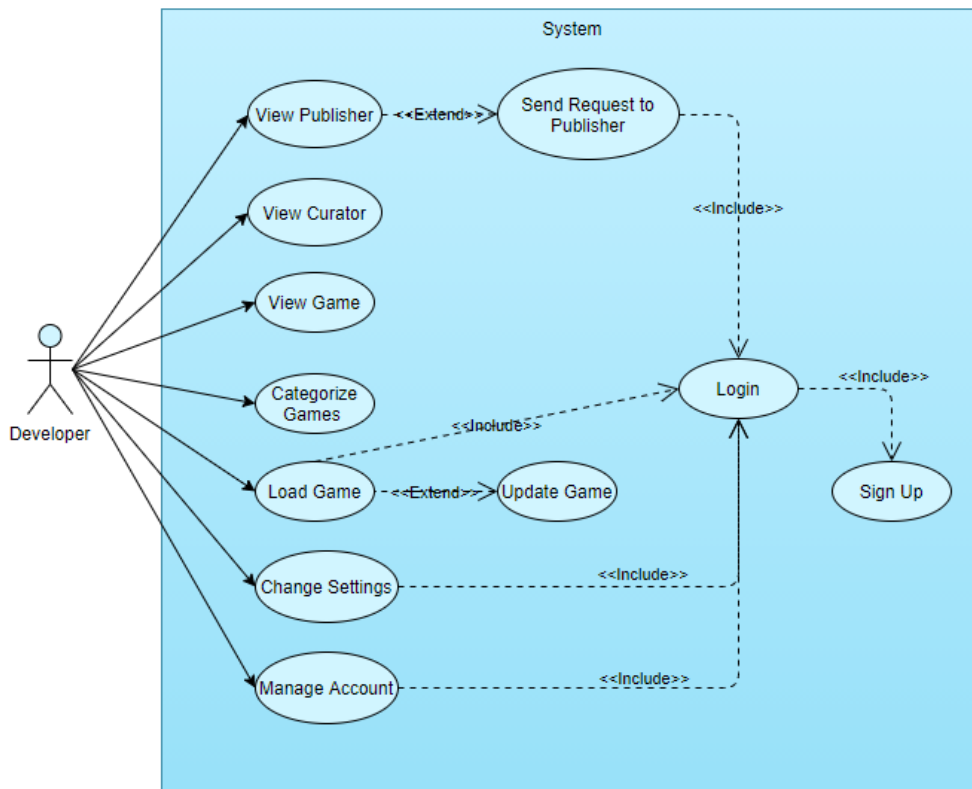


Figure 4 : Developer Use Case Diagram

4.1.4 Publisher

Sign Up: Publishers use their email, username, name, surname, age and passwords to create an account on the system. After they choose the Publisher as their account type, their account is created. In order to purchase, play and rate the games, add friends and do other account based actions on the system, they have to have an account. However, there are a couple of features that they can do without creating an account, such as viewing games, comments of the games and publishers.

Login: Publishers login to the access to their accounts by using email addresses and the password saved in the database of the system. If they don't have an account, they can sign up and create their account.

View Publisher: Publishers view other publishers' profiles and view the information about the games that they publish.

View Curator: Publishers view curators' profiles and view the latest comment that they wrote to the games.

View Game: Publishers view the games. They can search and categorize them. When they view a game, they can access the information such as release date, version, system requirements of the game. They also can view the modes of the game, curator and player rates and other comments. Unlike Players and Curators in the system, they cannot write comments to the games.

View Developer: Publishers can view developers' profiles and check the uploaded games and updates.

View Curator: Publishers view curators' profiles and view the latest comment that they wrote to the games.

Publish Game: Publishers accept or decline the game or update requests that developers have sent to them. They can change the description, game name and other specifications that developers wrote for the game.

Determine Cost: Publishers determine the cost of the games when they are publishing the game.

Make Discount: Publishers make discounts on the games and they can determine the discount period as limitless or limited time discounts.

Notify Players: Publishers notify players when they publish a new game or new update.

Manage Account: Publishers manage their account by changing the email, passwords, username, name and surname.

Change Settings: Publishers change website colors according to their personal preferences.

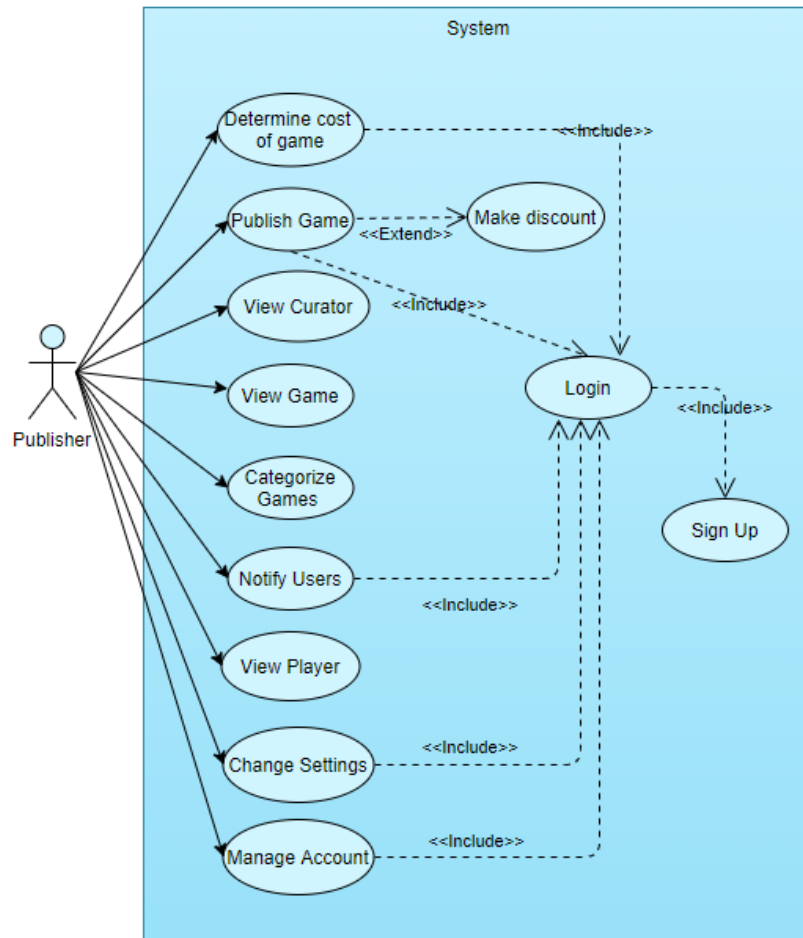


Figure 5: Publisher Use Case Diagram

4.1.5 Curator

Sign Up: Curators use their email, username, name, surname, age and passwords to create an account on the system. After they choose the Curator as their account type, their account is created. In order to purchase, play and rate the games, add friends and do other account based actions on the system, they have to have an account. However, there are a couple of features that they can do without creating an account, such as viewing games, comments of the games and publishers.

Login: Curators login to the access to their accounts by using email addresses and the password saved in the database of the system. If they don't have an account, they can sign up and create their account.

View Followers: Curators view their followers and the games that they are playing.

View Game: Curators view the games. They can search and categorize them. When they view a game, they can access the information such as release date, version, system requirements of the game. They also can view the modes of the game, curator and player rates and other comments.

Write Comment: Curators write comments to the games. Their comments are published on the game page that is associated with the comment. Their followers get notified when they write a new comment.

Rate Game: Curators rate the games by giving from 1 to 5 stars to the game and the average is calculated by the stars that all other users of the system have given. This calculation is added to the game pages to help others to review the rate of the games.

Manage Account: Curators manage their account by changing the email, passwords, username, name and surname.

Change Settings: Curators change website colors according to their personal preferences.

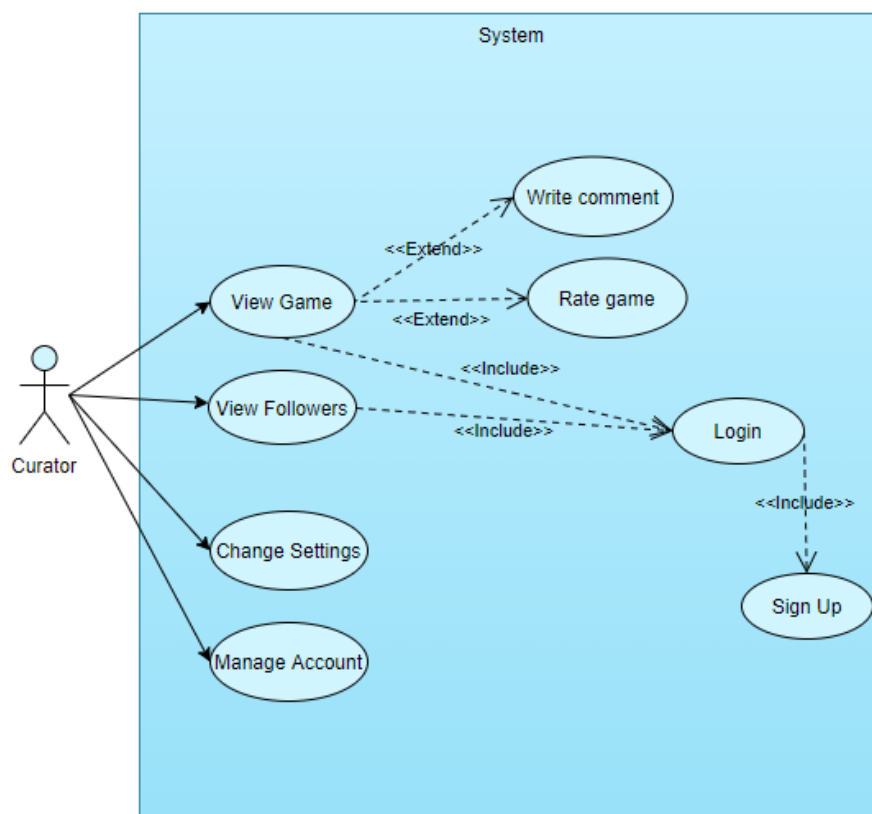


Figure 6 : Curator Use Case Diagram

4.2 Algorithms

4.2.1 Displaying Game Algorithms

All of the user accounts in the system can view the games. This system has the curators and publishers to help players to have idea of the games and make their selection easier so that they can find the games they would like to play. The algorithm has to be well designed to choose and show the games in a good order for users to purchase them. Our priority to display the games is going to be based on the top rated games according to the game rates. Other than game rates, the games are going to be displayed in an order that curators that players' follow have rated. In addition to curators and top rankings, users are going to display the games in an order that they friends have.

4.2.2 Login Requirements

All of the user accounts are going to sign up with their email addresses, passwords and usernames. In order to prevent the system from the not used accounts and have them in our databases as trash, the system requires accounts with unique emails so that users cannot sign up more than one time with one email.

After signing up, the system checks if the email and the password is saved to the database and checks whether they match with each other or not.

4.3 Data Structures

The relation schemas we created uses Numeric Types, String Types and Char Types.

Numeric types are used in order to store numeric data like user_id, level, pub-id and points. Using numeric types, INT, MEDIUMINT, TINYINT and BIT are used to decrease space usage.

String and char types are using storage of characters such as pub-name, pub-address and game-name. String and Char types also include VARCHAR and CHAR. So, we can store the characters in various ways.

5 User Interface Design and Corresponding SQL Statements

5.1 Sign In / Login

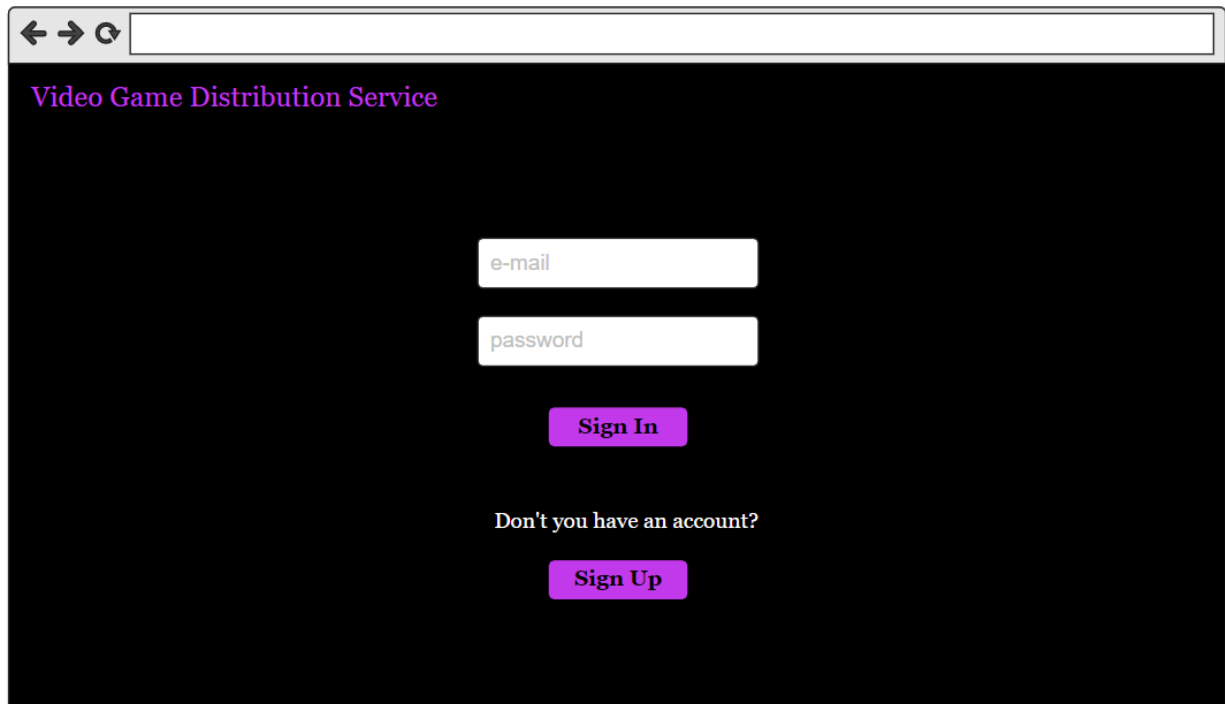
A screenshot of a web browser window displaying the login interface for the 'Video Game Distribution Service'. The browser's address bar is empty. The page has a black background. At the top left, the text 'Video Game Distribution Service' is written in a purple font. In the center, there are two white input fields: the top one is labeled 'e-mail' and the bottom one is labeled 'password'. Below these fields is a purple button with the text 'Sign In' in white. Underneath the button is the text 'Don't you have an account?' in white. At the bottom, there is another purple button with the text 'Sign Up' in white.

Figure 7 : Log In Interface

Inputs: @email, @password

Process: All types of accounts will enter the system by providing their email address and password. We will determine the type of the account by email address and in which sub account table it is also presented. If the provided information is not valid a pop-up screen will be shown to users.

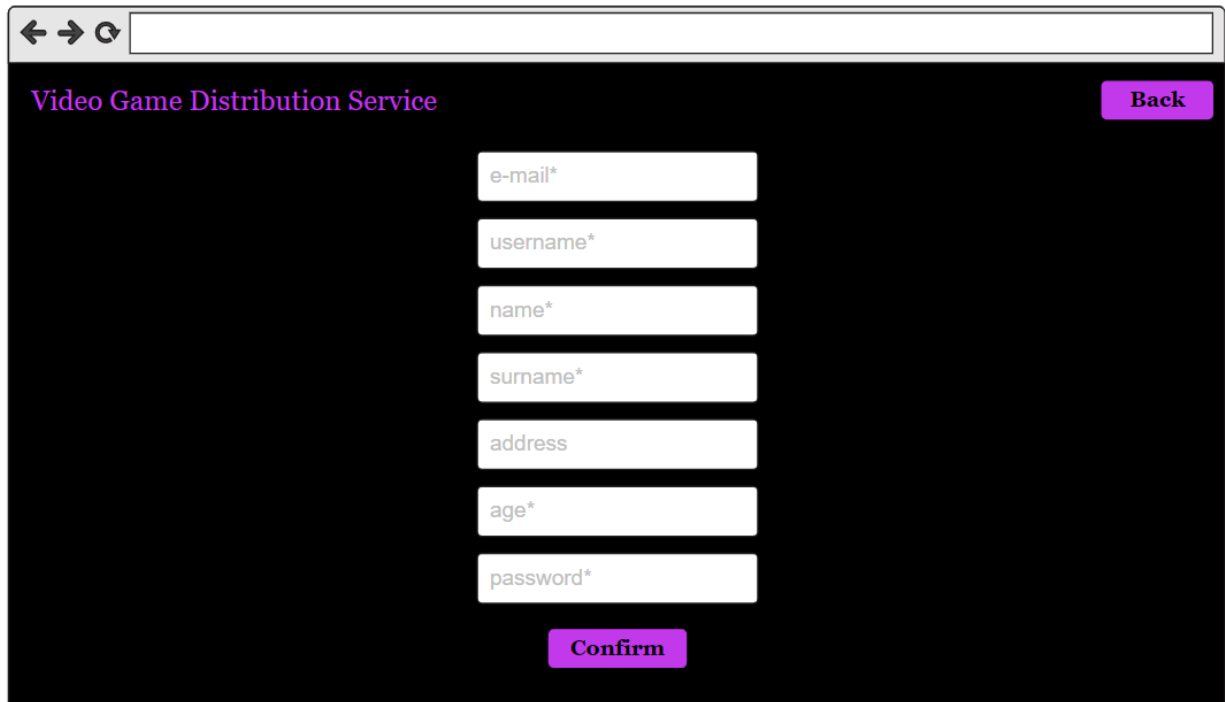
SQL Statements:

Select *

From Account

WHERE email = @email AND password = @password

5.2 Sign Up



The image shows a web browser window with a dark blue background. The title bar of the browser is at the top. Below the title bar, the text "Video Game Distribution Service" is displayed in a light blue font on the left, and a "Back" button is on the right. In the center, there is a vertical stack of seven white input fields. The first six fields are labeled "e-mail*", "username*", "name*", "surname*", "address", and "age*" from top to bottom. The seventh field is labeled "password*". Below these fields is a "Confirm" button.

Figure 8: Registration Interface

Inputs: @email, @username, @name, @surname, @address, @age, @password

Process: If a user does not have an account yet he/she can enroll by providing necessary information. After the confirm button clicked the user will be navigated to the next page that will make him/her choose an account type.

SQL Statements:

```
INSERT INTO Account
```

```
VALUES (@email, @username, @name, 0, @surname, @address, @age)
```

5.3 Selecting Account Type

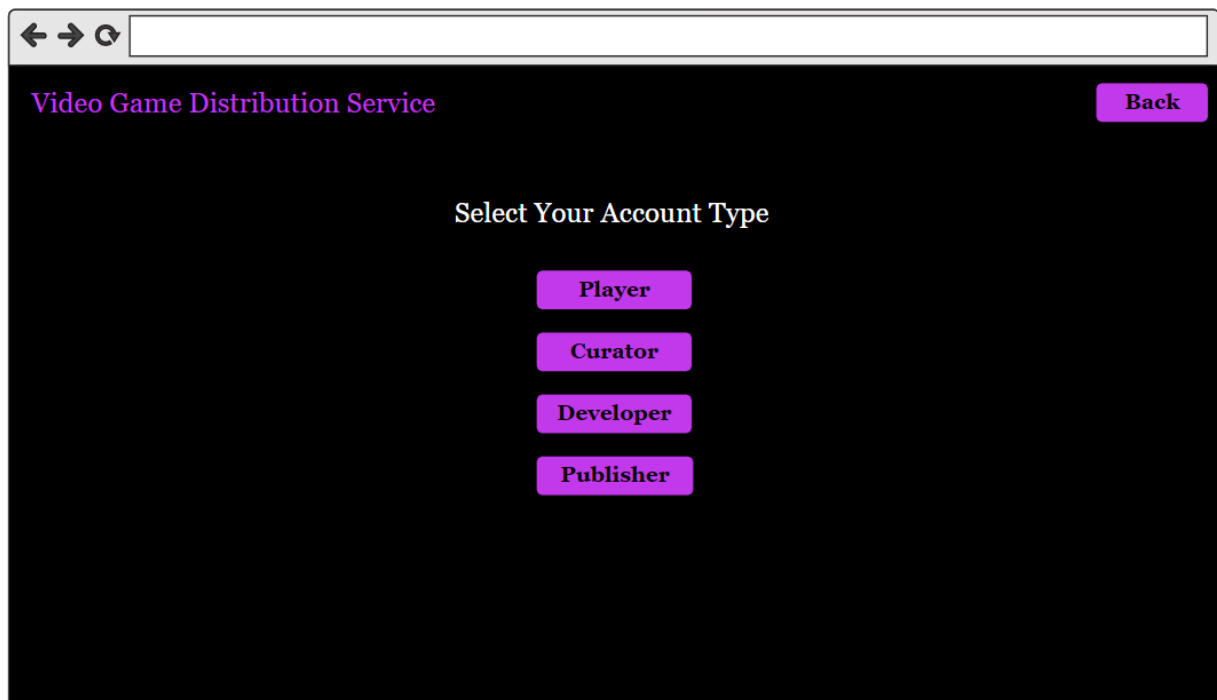


Figure 9: Account Type Interface

Inputs: @accountType

Process: As mentioned in the previous section, the user will select his/her account type by clicking on the account type. We will then insert the related email into the corresponding account type table. However, To become a developer and publisher we require additional information. This information will be taken by showing a new page right after clicking on one of these buttons. The UI's for that case are not given to protect simplicity.

The unique user-id, cur-id and pub-id for entities will be determined by making calculations programmatically. We will use specific algorithms and search methods to assign each type of account a unique id.

5.4 Main Page for Users

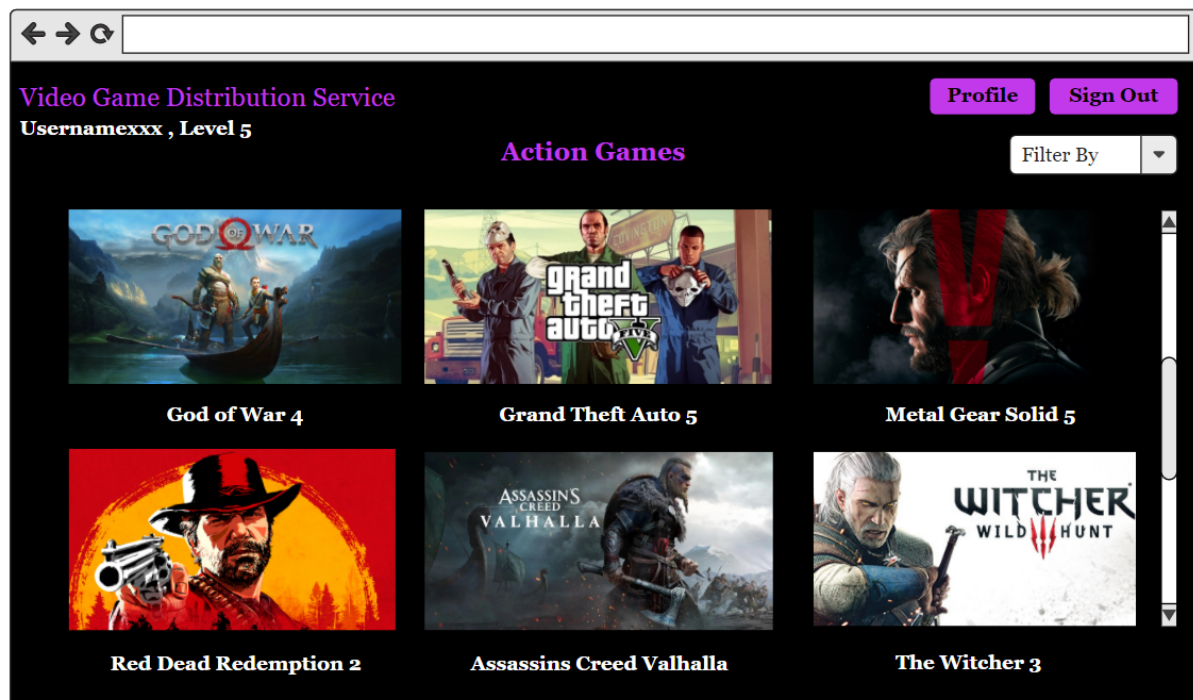


Figure 10 : Display Games Interface

Inputs: @filterValue

Process: Users will see this page immediately after logging in the system. By default the top rated games will be shown, and users will be able to change the filter type. Thus, the whole game market will be shown here for users. As an example, the case of selecting filters by action games is given.

SQL Statements:

```
SELECT game-id, game-name
FROM Game
WHERE genre = 'Action'
```

5.5 Game View for Users

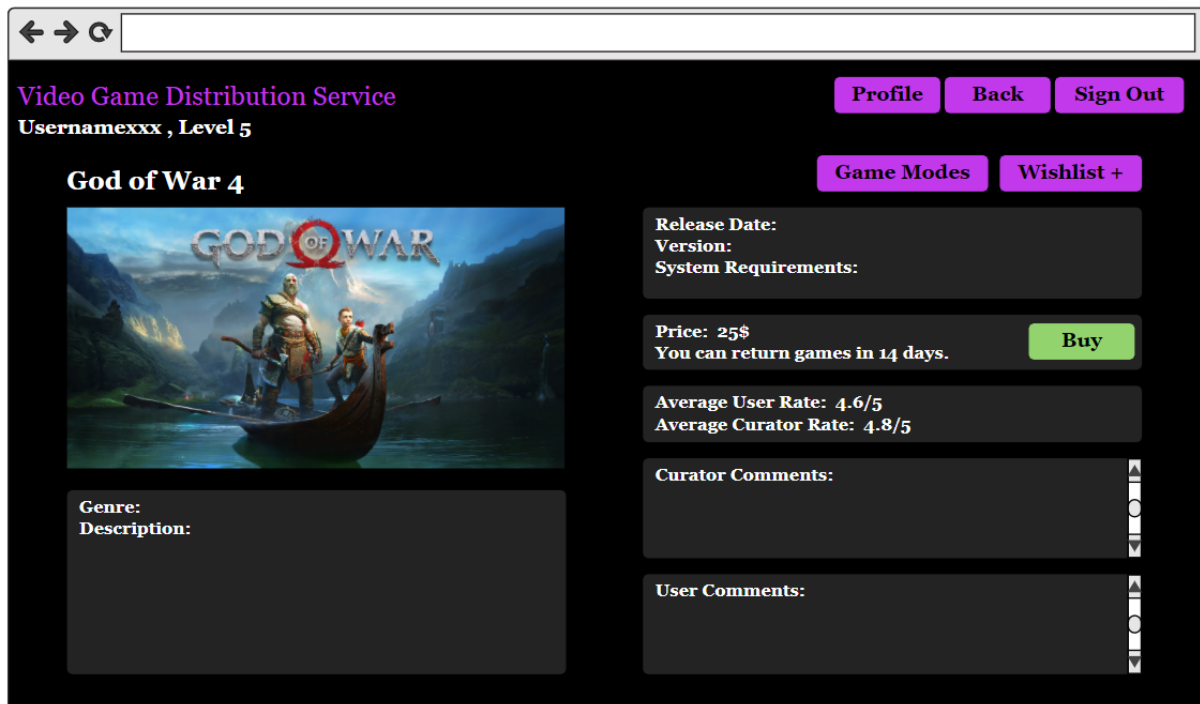


Figure 11 : Game Page Interface

Variable Input: @selectedGameId

Process: When a user clicks on a game from the game store shown on the main page, then he/she will be directed into a page in which the detailed information about the game is provided. The fields are left blank for illustration. If the user wants to get the game then the Buy button should be pressed. For this case the @selectedGameId is assigned from the previous page which corresponds to the game-id of the game which is clicked.

SQL Statement:

Showing Game Information Rather Than Average Rates:

```
SELECT game-name, genre, content, version, release-date, price, requirements
FROM Game
WHERE game-id = @selectedGameId
```

Showing average rate information by computing:

```
WITH avg-rates(avg-cur-rate, avg-us-rate) AS
```

```

(SELECT avg(c-rate), avg(u-rate)
FROM user-rates NATURAL FULL OUTER JOIN cur-rates
WHERE game-id = @selectedGameId)
SELECT avg-cur-rate, avg-us-rate
FROM avg-rates

```

5.6 User Profile Showing Owned Games

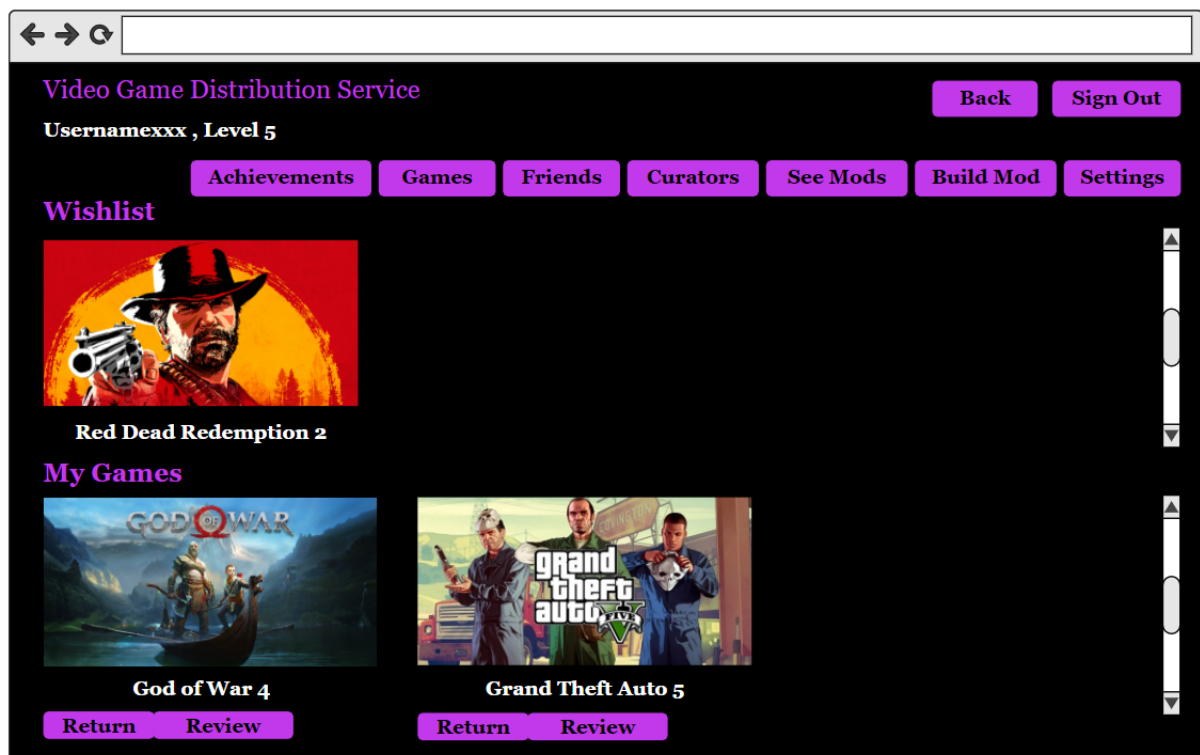


Figure 12 : Library Interface

Variable Input: @userId

Process: This page is shown for users after the “Profile” button is clicked in the Main Page. This page contains wishlisted games of the user and the downloaded games. Users may choose to return a game if the 14 day limit is satisfied. Pop up screen will show up when the return button is clicked asking for the user to confirm. Also, if a user clicks on the “Review” button, then a new page will open which is for giving rates and making comments to games. The similar case occurs for the Curators, so the SQL statements and the UI illustration can be seen in section 5.9. @userId is assigned when the user enters the system and it will be kept in a variable in the whole program according to session.

SQL Statements:

For showing wishlisted games:

```
SELECT game-name  
FROM wishlist NATURAL JOIN Game  
WHERE user-id = @userid
```

For showing downloaded games:

```
SELECT game-name  
FROM has-game NATURAL JOIN Game  
WHERE user-id = @userId
```

5.7 Built Mode Page for Users

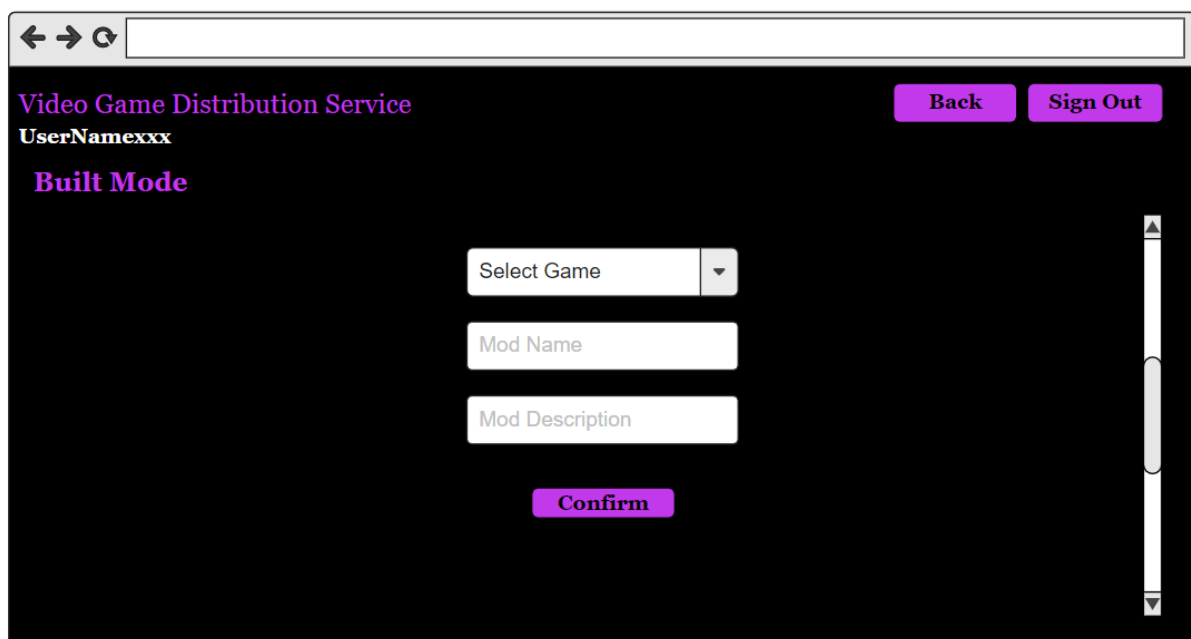


Figure 13 : Build Mode Interface

Inputs: @selectedGameId, @modName, @modDescription

Process: Users can build modes for specific games. After the “Build Mode” button is pressed from the users profile page, this page will be shown. Users can add mods only after providing required information.

@selectedGameId will be determined from the select game bar. In this bar the game name will be displayed by its publisher and release information that will help the user to distinguish between games. Later by using the selected values game-id will be found and assigned to this variable again.

@mod-id will be calculated in the programming side. We will assign unique numbers for each mode.

SQL Statements:

INSERT INTO Mod

VALUES @mod-id, @modName, @modDescription, @selectedGameId

5.8 Downloaded Modes of User

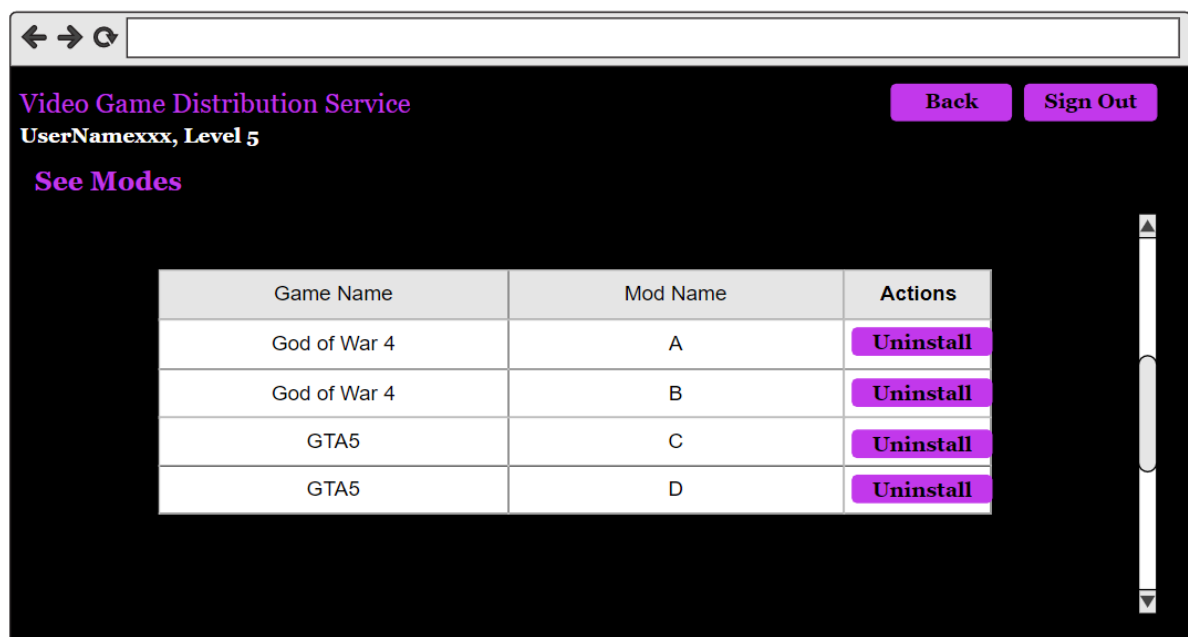


Figure 14 : Build Library Interface

Process: In this page users will be able to see their downloaded mods. The mods will be displayed in a table with the corresponding information. Users will be able to uninstall their mods from the games by pressing the “Uninstall” button.

@userId determined when the session for that user started.

SQL Statements:

To show the table:

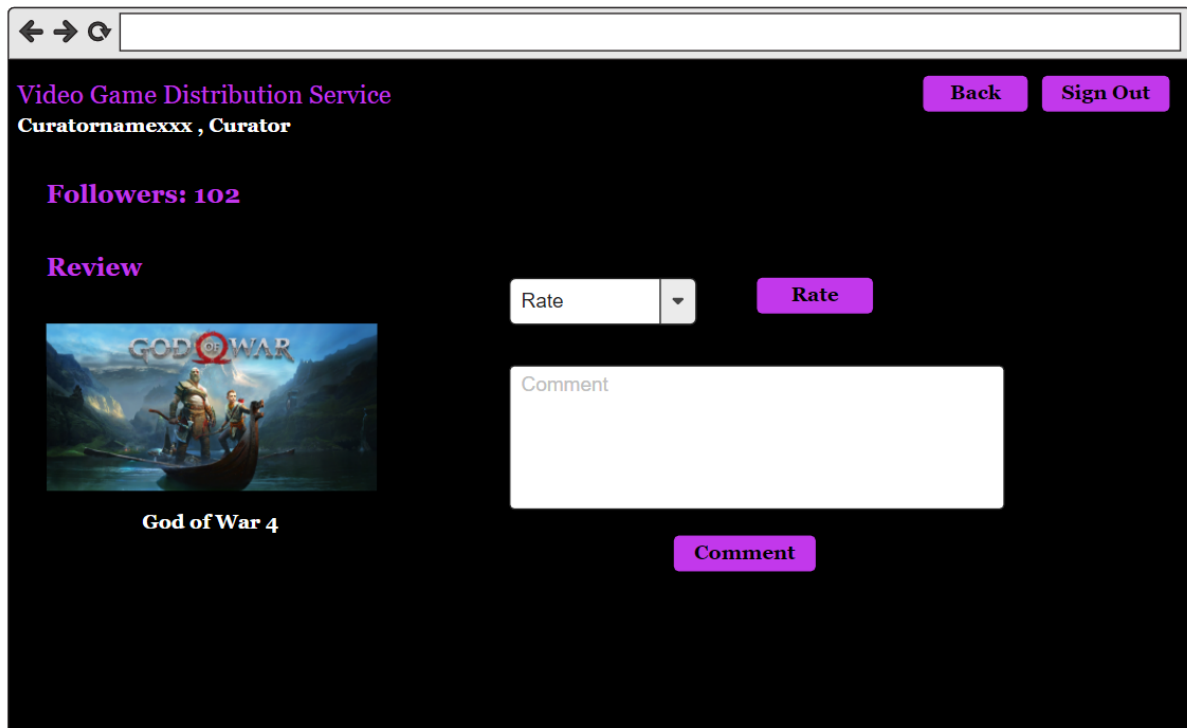
```
SELECT game-name, mod-name  
FROM has-game NATURAL JOIN Game NATURAL JOIN Mod NATURAL JOIN  
has-user-mode  
WHERE user-id = @userId
```

@selectedModName, and @selectedGameName will be determined when the corresponding “Uninstall” button is clicked.

To Uninstall:

```
DELETE FROM has-user-mod  
WHERE mod-id IN  
(SELECT mod-id  
FROM has-user-node NATURAL JOIN Mod NATURAL JOIN Game  
WHERE game-name = @selectedGameName AND user-id = @userId AND mod-name =  
@selectedModName)
```

5.9 Comment and Rate by a Curator



Video Game Distribution Service
Curatornamexxx, Curator

Back Sign Out

Followers: 102

Review

God of War 4

Rate

Rate

Comment

Comment

Figure 15 : Write Comment Interface

Inputs: @curatorRate, @curatorComment

Process: This page is for making comments for games and rating them. This example is for curator accounts but the same functionality appears for users also. To protect simplicity, a Curator version is given as an example.

@gameId will be assigned by the previous page where the game is clicked.

@curId is assigned when the session for that curator is started.

@curatorCommentId is assigned by making calculations to assign a unique id to comment in the programming side.

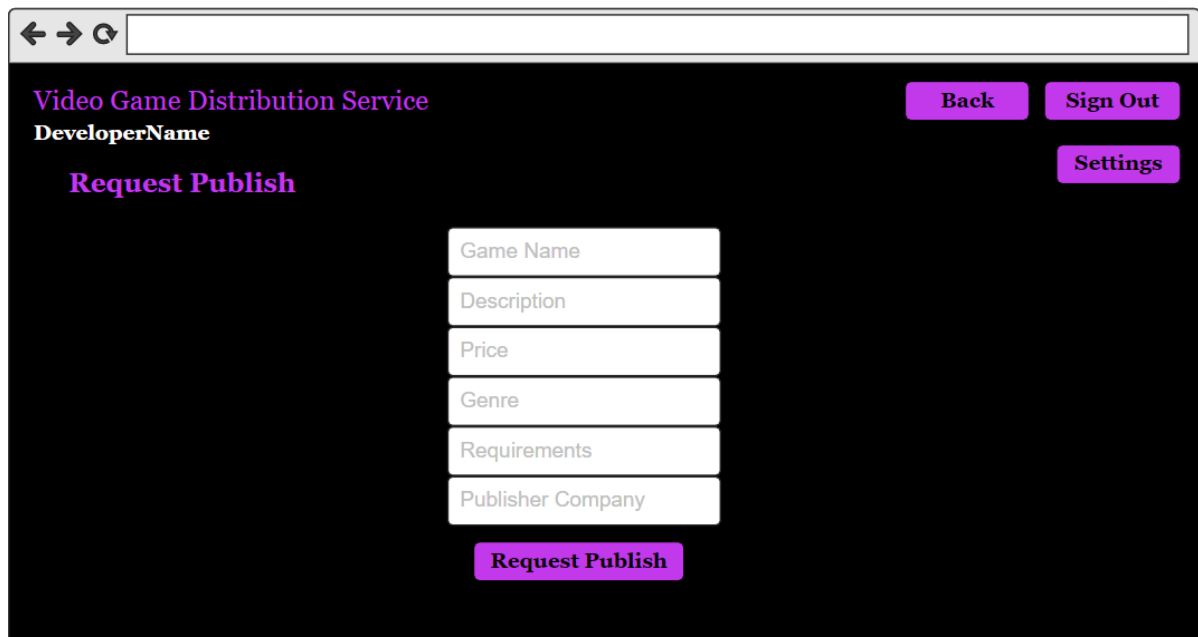
SQL Statements:

```
INSERT INTO curator-rates  
VALUES (@curId, @game-id, @curatorRate)
```

INSERT INTO Curator-Comment

VALUES (@curId, @curatorCommentId, @game-id, @curatorComment, @game-id)

5.10 Developers Ask for Publish



Video Game Distribution Service

DeveloperName

Request Publish

Back Sign Out Settings

Game Name

Description

Price

Genre

Requirements

Publisher Company

Request Publish

Figure 16 : Sending Request Interface

Inputs: @newGameName, @newDescription, @newPrice, @newGenre, @newReq

Process: Developer companies will ask for Publisher companies to publish their games. If a publisher is rejected they will be notified and they will be able to make a new request for a different Publisher Company.

First by using game-id the new values for that game will be entered into Game relation. If the publication is rejected the tuple values will be set to null as before. This means that the game exists but not published yet. Thus, if a game is not in the publish relation and the values rather than game-id and game-name are null then it means that developer develops a game but that game is not published.

@devId is assigned from the developers session.

@pub-id will be found by the pub-name.

SQL Statements:


```
INSERT INTO ask  
VALUES (@devId, @pubId, 0)
```

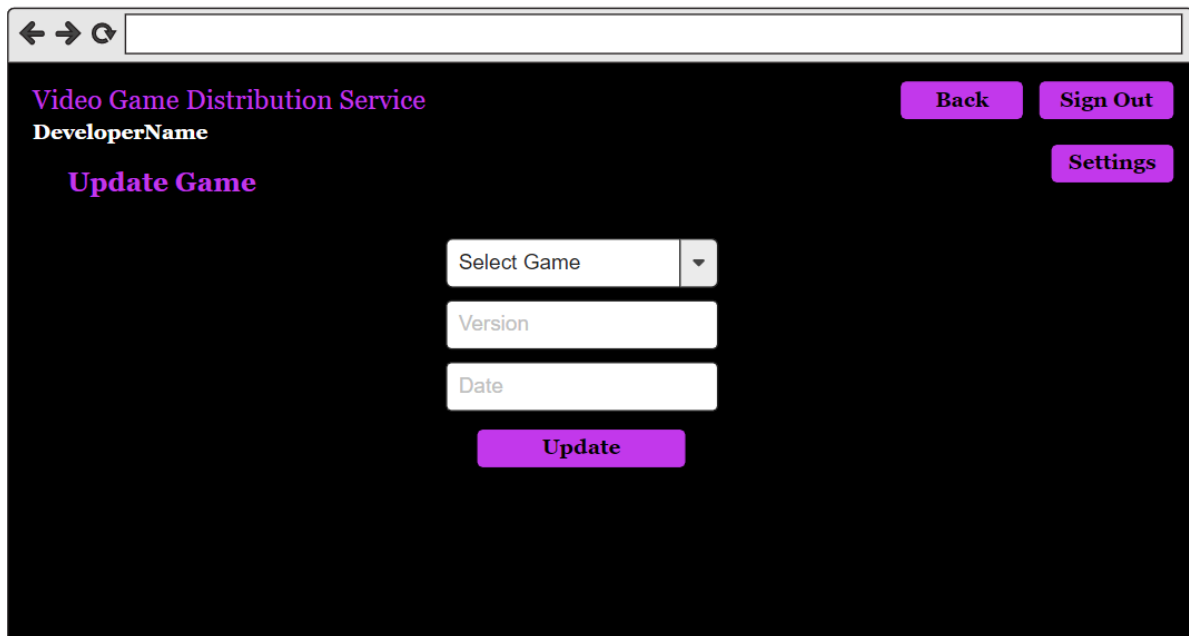
```
UPDATE Game  
SET game-description = @newDescription  
WHERE game-id =  
(SELECT game-id  
FROM develops, Game  
WHERE develops.game-id = Game.game-id)
```

```
UPDATE Game  
SET price = @newPrice  
WHERE game-id =  
(SELECT game-id  
FROM develops, Game  
WHERE develops.game-id = Game.game-id)
```

```
UPDATE Game  
SET genre = @newGenre  
WHERE game-id =  
(SELECT game-id  
FROM develops, Game  
WHERE develops.game-id = Game.game-id)
```

```
UPDATE Game  
SET requirements = @newRequirements  
WHERE game-id =  
(SELECT game-id  
FROM develops, Game  
WHERE develops.game-id = Game.game-id)
```

5.11 Developers Updating a Game



The screenshot shows a web browser window with a dark background. The browser's address bar is empty. The page header includes the text "Video Game Distribution Service" and "DeveloperName" on the left, and three blue buttons labeled "Back", "Sign Out", and "Settings" on the right. The main heading is "Update Game". Below this heading, there are three input fields stacked vertically: "Select Game" (a dropdown menu), "Version", and "Date". At the bottom of these fields is a blue button labeled "Update".

Figure 17 : Update Game Interface

Inputs: @selectedGameId, @newVersion, @newDate

Process: Developers will be able to update their games from this page.

SQL Statements:

```
INSERT INTO updates
```

```
VALUES (@selectedGameId, @devId, @newVersion, @newDate)
```

5.12 Publisher Company Viewing Publish Requests

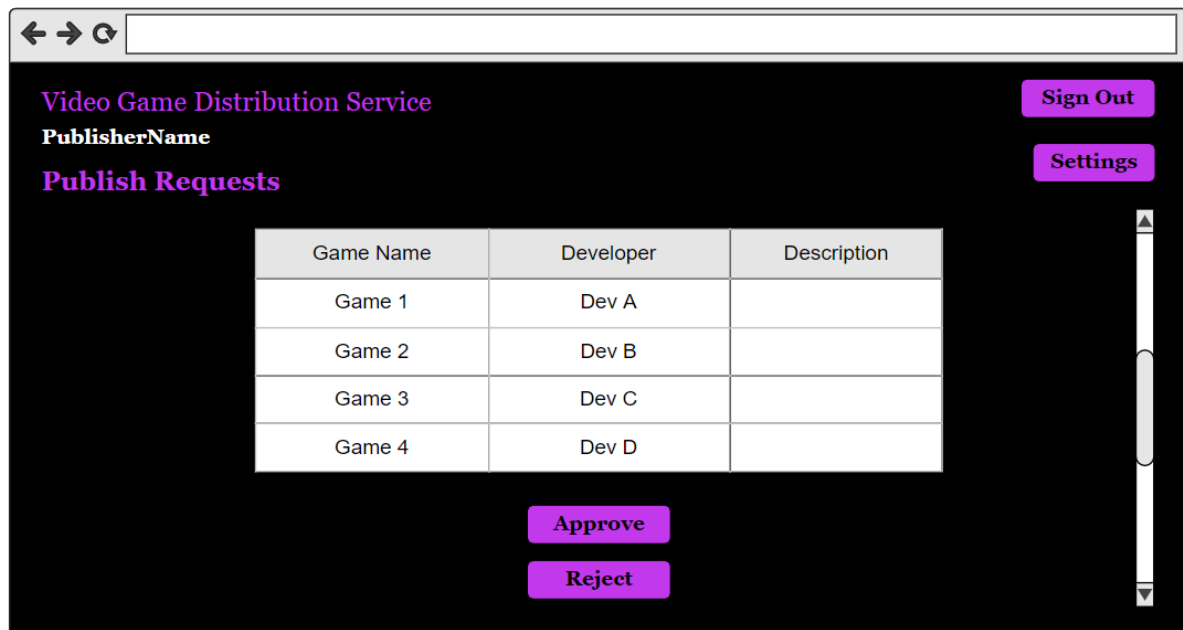


Figure 18: Publish / Decline Interface

Process: Publishers can view the requests that are made to them. They can accept or reject requests by selecting the corresponding row and then pressing the corresponding button.

SQL Statements:

To show the table:

```
SELECT game-name, dev-id, game-description  
FROM Game NATURAL JOIN develops
```

@pubId is assigned when the session for that publisher is assigned.

If approved:

```
INSERT INTO publish  
SELECT pub-id, game-id  
FROM develops NATURAL JOIN ask NATURAL JOIN Game  
WHERE pub-id = @pubId
```

5.13 Extra Functionality: Achievements

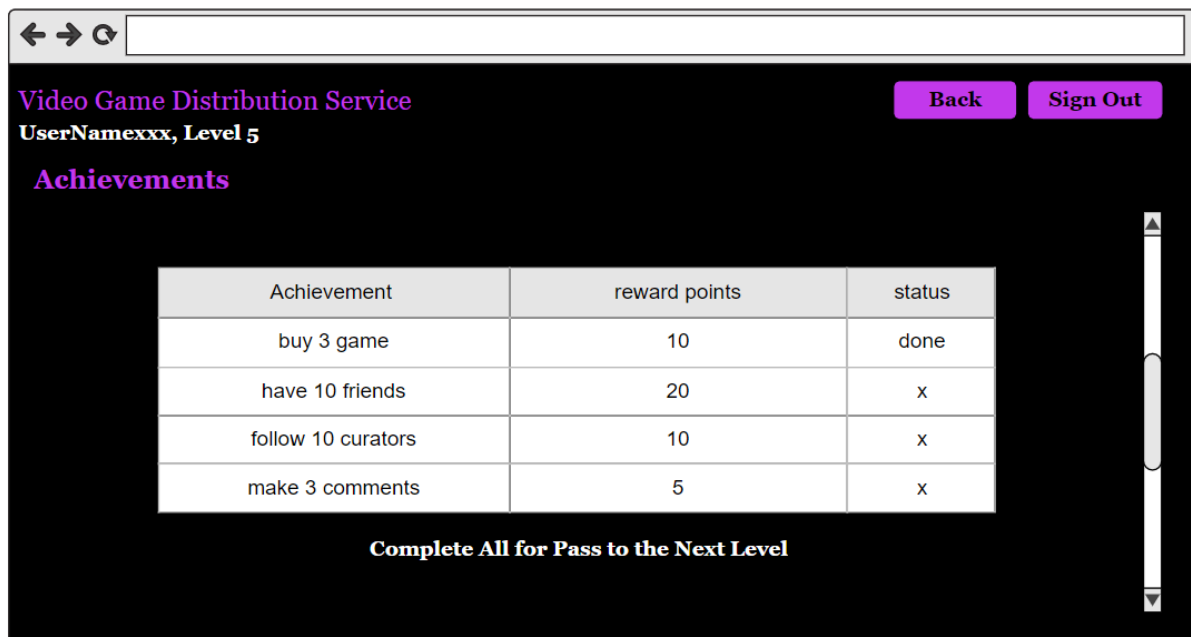


Figure 19 : Achievements Interface

Process: This page is for our extra functionality which is giving achievement points for users. Users will pass levels by completing each achievement in the Achievements table. For each level the achievement contents will be changed. After completing a level users will earn credits and they will be able to buy games with those credits.

@userId is assigned when the session for that user is assigned.

SQL Statements:

For showing table contents:

```
SELECT A.ach-name, A.ach-reward
FROM Achievement A
WHERE A.ach-id IN (SELECT ach-id
                  FROM has-achievement H, Achievement A2
                  WHERE H.level-id = A2.level-id)
```

SELECT case

```

WHEN @userId IN (SELECT user-id
                  FROM has-achievement H, Achievement A2
                  WHERE @userId = H.user-id AND H.level-id = A.level-id)
THEN 'done'
ELSE 'X'

```

6 Advanced Database Components

In this section we show the advanced database components of our project.

6.1 Views

Top 10 cheapest game:

```

CREATE VIEW top_10 AS
SELECT game-name
FROM Game
ORDER BY price DESC
LIMIT 10

```

Games with a particular genre:

```

CREATE VIEW genre AS
SELECT game-name
FROM Game
WHERE genre = @genre

```

Games with a particular publisher:

```

CREATE VIEW publisher-games AS
SELECT game-name,
FROM Game NATURAL JOIN Publisher
WHERE pub-name = @pub-name

```

6.2 Stored Procedures

- Log in and sign up procedures will be stored so we can check for validity during login.

6.3 Triggers

- When a new game is added to the system, the related top 10 cheapest game list can change.
- If a user or a curator rates a game, the rating of the game can be updated.

6.4 Constraints

1. The system cannot be used without an account.
2. The system requires at least a valid email address, username, name, surname, age and a password to be enrolled.
3. An unapproved publisher cannot publish game
4. The password cannot be shorter than 6 characters and cannot be longer than 15 characters
5. Users cannot rate and comment the same game more than one time, but they can edit
6. Curators cannot rate and comment the same game more than one time, but they can edit
7. Each game cannot have more than one publisher
8. A user can add a user only one time
9. A mod can be developed by only one user
10. Users can have only one unique preference which means preferences cannot change device by device.
11. There will be only one payment method which is an application wallet. Users can add money to their wallet and then buy games with money in that wallet.

7 Implementation Plan

For our system functionalities and user interface in our video game distribution system, we are planning to use HTML, CSS, JavaScript, and PHP. In order to manage the flow of data in our project, we are planning to use MySQL.