Bilkent University

Department of Computer Engineering

# Database Systems Project

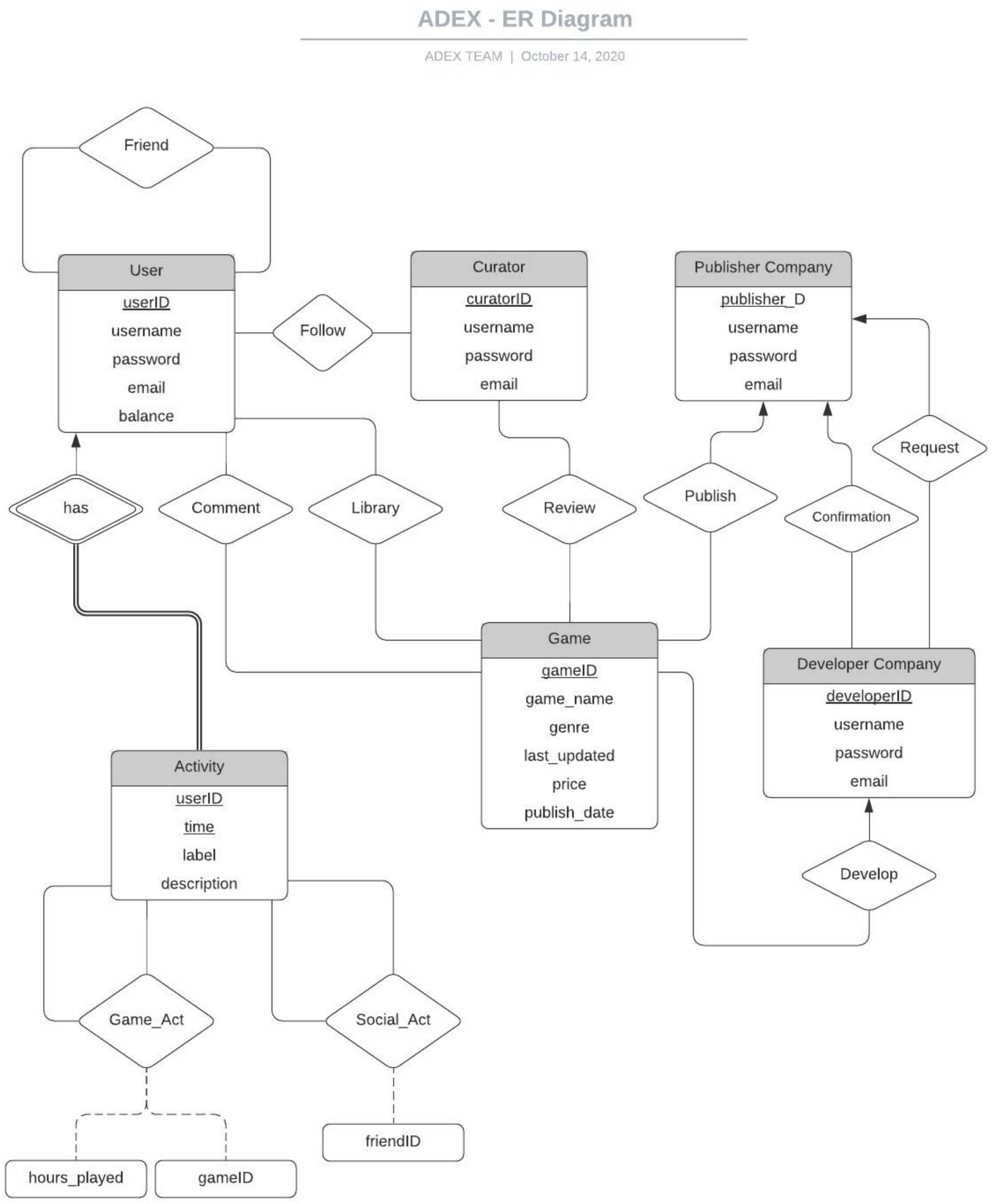
CS - 353 - Section 1

**Video Game Digital Distribution Service - Adex**
**Project Design Report**

Olcay Akman - 21702671
Muhammet Kamil Gök - 21600879
Ege Hakan Karaağaç - 21702767
Rumeysa Özaydın - 21601558

# 1 Revised E/R Model

Our ER model from the previous report, without the modifications, is given below.



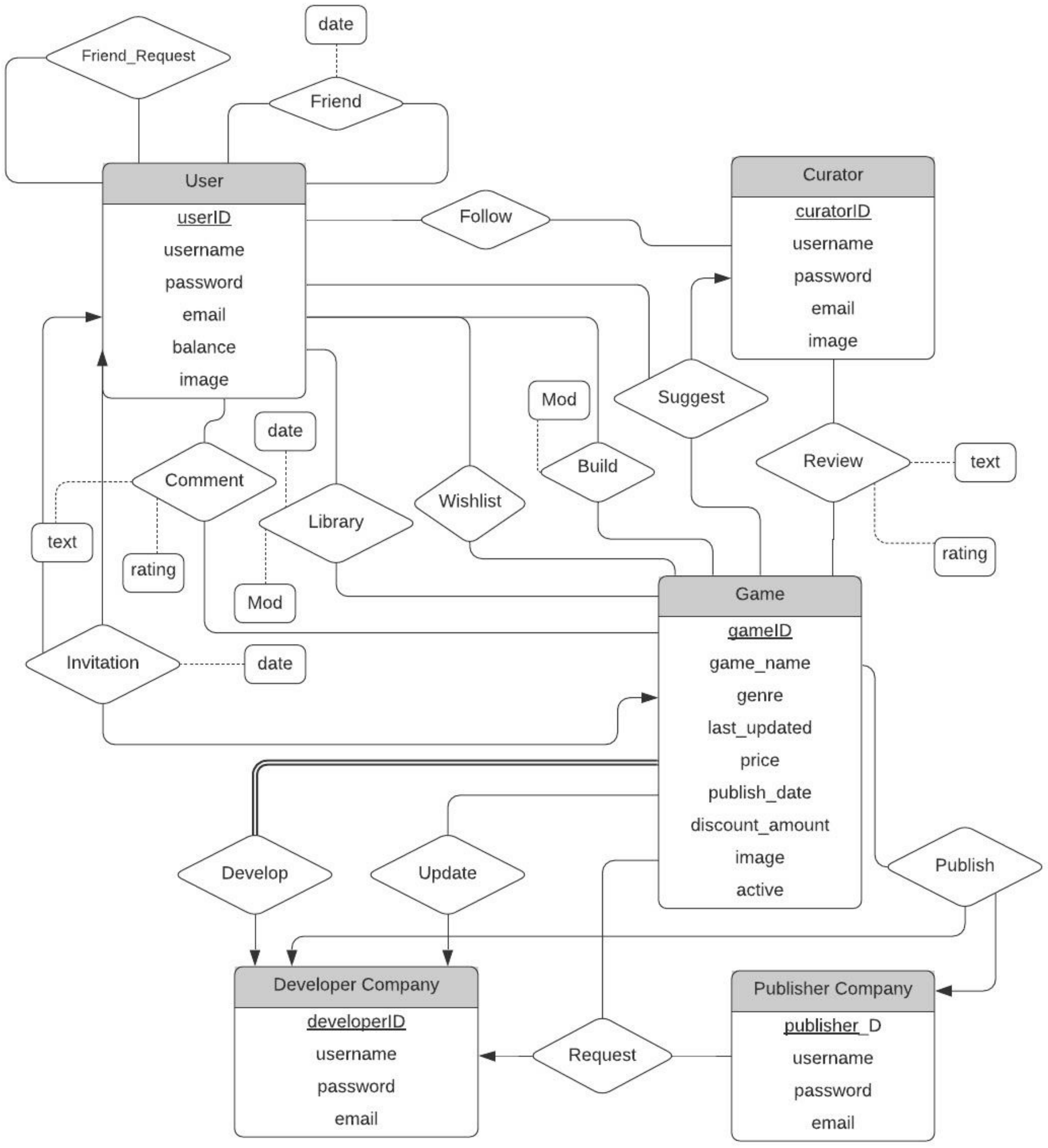
We have made the following changes on our ER model based on the feedback of our TA.

- We have deleted the weak Activity entity and its relations. Instead we decided it can be done using views.
- We have added new attributes “text”  and “rating” to review and comment relations.
- We have made the relations “Request” and “Publish” between “Developer Company”, “Publisher Company” and “Game entities” in order to make publishing requests and publishing games.
- We have added an “Update” relation between “Game” and “Developer Company” entities.
- We have added a “Build” relation between “Game” and “User” entities which has an attribute “mod” in order to build a mod for a game.
- We have added a “Friend_Request” relation between Users to get permission from users before adding them as friends.
- As a new functionality, we have added an option for users to invite friends to games. For this functionality, we have added an “Invitation” relation between “User” and “Game” entities.
- As a new functionality, we have added a wishlist page for users. For this functionality, we have added a “Wishlist” relation between “User” and “Game” entities. Users will get notified when a game in their wishlist has a discount.
- As a new functionality, we have added an option for curators to suggest a game to their followers. For this functionality, we have added a “Suggest” relation between “User”, “Game” and “Curator” entities.
- In the relation “Develop” between “Game” and “Developer Company” entities, we have made “Game” an entity to have a total participation. Because all games must be developed by a developer company. Games have an attribute called active and it is “No” until they are published. When they are not active, they can not be seen by Users in the store.

Below is the *revised* version of our ER diagram, with the changes specified above.

## ADEX - ER Diagram

ADEX TEAM | November 23, 2020



# 2 Relational Schemas

## 2.1 Relational Schemas

### 2.1.1 User

**Relational Model:**

user(userID, username, password, email, balance, image)

**Functional Dependencies:**

userID -> username, password, email, balance, image

username -> userID, password, email, balance, image

email -> userID, username, password, balance, image

**Candidate Key:**

{(userID), (username), (email)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE user (
      userID INT NOT NULL AUTO_INCREMENT,
      username VARCHAR(20) NOT NULL,
      password VARCHAR(40) NOT NULL,
      email VARCHAR(120) NOT NULL,
      balance DECIMAL(8,2) NOT NULL DEFAULT 0,
      image VARCHAR(120),
      PRIMARY KEY (userID),
      UNIQUE (userID, username, email)
);
```

### 2.1.2 Friend

**Relational Model:**

friend(userID1, userID2, date)

**Functional Dependencies:**

userID1, userID2 -> date

**Candidate Key:**

{(userID1, userID2)}

**Normal Form**:

BCNF

**Table Definition:**

```
CREATE TABLE friend (
      userID1 INT NOT NULL,
      userID2 INT NOT NULL,
      date DATETIME,
      PRIMARY KEY (userID1, userID2),
      FOREIGN KEY (userID1) REFERENCES user
                           ON DELETE CASCADE
                           ON UPDATE CASCADE,
      FOREIGN KEY (userID2) REFERENCES user
                           ON DELETE CASCADE
                           ON UPDATE CASCADE
);
```

### 2.1.3 Friend Request

**Relational Model:**

friend_request(<u>userID1, userID2</u>)

**Functional Dependencies:**

None

**Candidate Key:**

{(userID1, userID2)}

**Normal Form**:

BCNF

**Table Definition:**

```
CREATE TABLE friend_request (
        userID1 INT NOT NULL,
        userID2 INT NOT NULL,
        PRIMARY KEY (userID1, userID2),
        FOREIGN KEY (userID1) REFERENCES user
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (userID2) REFERENCES user
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

### 2.1.4 Curator

**Relational Model:**

curator(<u>curatorID</u>, username, password, email, image)

**Functional Dependencies:**

curatorID -> username, password, email, image

username -> curatorID, password, email, image

email -> curatorID, username, password, image

**Candidate Key:**

{(curatorID), (username), (email)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE curator (
        curatorID INT NOT NULL AUTO_INCREMENT,
        username VARCHAR(20) NOT NULL,
        password VARCHAR(40) NOT NULL,
        email VARCHAR(120) NOT NULL,
        image VARCHAR(120),
        PRIMARY KEY (curatorID),
        UNIQUE (curatorID, username, email)
  );
```

### 2.1.5 Follow

**Relational Model:**

follow(<u>userID, curatorID</u>)

**Functional Dependencies:**

None

**Candidate Key:**

{(userID, curatorID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE follow(
      userID INT NOT NULL,
      curatorID INT NOT NULL,
      PRIMARY KEY (userID, curatorID),
      FOREIGN KEY (userID) REFERENCES user
                           ON DELETE CASCADE
                           ON UPDATE CASCADE,
      FOREIGN KEY (curatorID) REFERENCES curator
                           ON DELETE CASCADE
                           ON UPDATE CASCADE
);
```

### 2.1.6 Game

**Relational Model:**

game(<u>gameID</u>, game_name, genre, last_updated, price, discount_amount, image, active)

**Functional Dependencies:**

gameID -> game_name, genre, last_updated, price, discount_amount, image, active

game_name -> gameID, genre, last_updated, price, discount_amount, image, active

**Candidate Key:**

{(gameID), (game_name)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE game(
      gameID INT NOT NULL AUTO_INCREMENT UNIQUE,
      game_name VARCHAR(20) NOT NULL UNIQUE,
      genre VARCHAR(20),
      last_updated DATETIME,
      price DECIMAL(6,2) NOT NULL,
      image VARCHAR(120),
      active VARCHAR(20),
      discount_amount DECIMAL(2,2) NOT NULL DEFAULT 0,
      PRIMARY KEY (gameID)
);
```

### 2.1.7 Invitation

**Relational Model:**

invitation(<u>userID1, userID2, gameID, date</u>)

**Functional Dependencies:**

None

**Candidate Key:**

{(userID1, userID2, gameID, date)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE invitation (
	userID1 INT NOT NULL,
	userID2 INT NOT NULL,
	gameID INT NOT NULL,
	date DATETIME,
	PRIMARY KEY (userID1, userID2, gameID, date),
	FOREIGN KEY (userID1) REFERENCES user
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (userID2) REFERENCES user
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (gameID) REFERENCES game
				ON DELETE CASCADE
				ON UPDATE CASCADE
);
```

### 2.1.8 Comment

**Relational Model:**

comment(<u>userID, gameID</u>, date, text, rating)

**Functional Dependencies:**

userID, gameID -> date, text, rating

**Candidate Key:**

{(userID, gameID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE comment(
	userID INT NOT NULL,
	gameID INT NOT NULL,
	date DATETIME,
        text LONGTEXT,
        rating INT NOT NULL
        PRIMARY KEY (userID, gameID),
        FOREIGN KEY (userID) REFERENCES user
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (gameID) REFERENCES game
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

### 2.1.9 Library

**Relational Model:**

library(<u>userID, gameID, mod</u>, date)

**Functional Dependencies:**

userID, gameID, mod -> date

**Candidate Key:**

{(userID, gameID, mod)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE library (
        userID INT NOT NULL,
        gameID INT NOT NULL,
        mod VARCHAR(20) NOT NULL,
        date DATETIME,
        PRIMARY KEY (userID, gameID, date),
        FOREIGN KEY (userID) REFERENCES user
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (gameID) REFERENCES game
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

### 2.1.10 Wishlist

**Relational Model:**

wishlist(userID, gameID)

**Functional Dependencies:**

None

**Candidate Key:**

{(userID, gameID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE wishlist(
      userID INT NOT NULL,
      gameID INT NOT NULL,
      PRIMARY KEY (userID, gameID),
      FOREIGN KEY (userID) REFERENCES user
                            ON DELETE CASCADE
                            ON UPDATE CASCADE,
      FOREIGN KEY (gameID) REFERENCES game
                            ON DELETE CASCADE
                            ON UPDATE CASCADE
);
```

### 2.1.11 Build

**Relational Model:**

build(userID, gameID, mod)

**Functional Dependencies:**

None

**Candidate Key:**

{((userID, gameID, mod)}

**Normal Form:**
BCNF

**Table Definition:**
```
CREATE TABLE build(
	userID INT NOT NULL,
	gameID INT NOT NULL,
	mod VARCHAR(20) NOT NULL,
	PRIMARY KEY (userID, gameID, mod),
	FOREIGN KEY (userID) REFERENCES user
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (gameID) REFERENCES game
				ON DELETE CASCADE
				ON UPDATE CASCADE
);
```

### 2.1.12 Suggest

**Relational Model:**
suggest(<u>curatorID, userID, gameID</u>)

**Functional Dependencies:**
None

**Candidate Key:**
{(curatorID, userID, gameID)}

**Normal Form:**
BCNF

**Table Definition:**
```
CREATE TABLE suggest(
	curatorID INT NOT NULL,
	userID INT NOT NULL,
	gameID INT NOT NULL,
	PRIMARY KEY (curatorID, userID, gameID),
	FOREIGN KEY (curatorID ) REFERENCES curator
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (userID) REFERENCES user
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (gameID) REFERENCES game
				ON DELETE CASCADE
				ON UPDATE CASCADE
);
```

### 2.1.13 Review

**Relational Model:**

review(curatorID, gameID, date, text, rating)

**Functional Dependencies:**

curatorID, gameID, date -> text, rating

**Candidate Key:**

{(curatorID, gameID, date)}

Normal Form:

BCNF

**Table Definition:**

```
CREATE TABLE review(
	curatorID INT NOT NULL,
	gameID INT NOT NULL,
	date DATETIME NOT NULL,
	text LONGTEXT,
	rating INT NOT NULL,
	PRIMARY KEY (curatorID, gameID, date),
	FOREIGN KEY (curatorID) REFERENCES curator
				ON DELETE CASCADE
				ON UPDATE CASCADE,
	FOREIGN KEY (gameID) REFERENCES game
	                     ON DELETE CASCADE
	                     ON UPDATE CASCADE
);
```

### 2.1.14 Publisher Company

**Relational Model:**

publisher_company(<u>publisherID</u>, username, password, email)

**Functional Dependencies:**

publisherID -> username, password, email

username -> publisherID, password, email

email -> publisherID, username, password

**Candidate Key:**

{(publisherID), (username), (email)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE publisher_company(
	publisherID INT NOT NULL UNIQUE,
	username VARCHAR(20) NOT NULL UNIQUE,
	password VARCHAR(40) NOT NULL,
	email VARCHAR(120) NOT NULL UNIQUE,
	PRIMARY KEY (publisherID)
);
```

### 2.1.15 Developer Company

**Relational Model:**

developer_company(developerID, username, password, email)

**Functional Dependencies:**

developerID-> username, password, email

username -> developerID, password, email

email -> developerID, username, password

**Candidate Key:**
{(developerID), (username), (email)}

**Normal Form**:
BCNF

**Table Definition:**
```
CREATE TABLE developer_company(
      developerID INT NOT NULL UNIQUE,
      username VARCHAR(20) NOT NULL UNIQUE,
      password VARCHAR(40) NOT NULL,
      email VARCHAR(120) NOT NULL UNIQUE,
      PRIMARY KEY (developerID)
);
```

### 2.1.16 Publish

**Relational Model:**
publish(publisherID, developerID, gameID, publish_date)

**Functional Dependencies:**
publisherID, developerID, gameID -> publish_date

**Candidate Key:**
{(publisherID, developerID, gameID)}

**Normal Form:**
BCNF

**Table Definition:**
```
CREATE TABLE publish(
      publisherID INT NOT NULL,
      developerID INT NOT NULL,
      gameID INT NOT NULL,
      publish_date DATETIME NOT NULL,,
      PRIMARY KEY (publisherID, developerID, gameID),
        FOREIGN KEY (publisherID ) REFERENCES publisher
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (developerID) REFERENCES developer
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (gameID) REFERENCES game
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

### 2.1.17 Update

**Relational Model:**

update(developerID, gameID, update_date, description)

**Functional Dependencies:**

developerID, gameID, update_date -> description

**Candidate Key:**

{(developerID, gameID, update_date)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE update (
        developerID INT NOT NULL UNIQUE,
        gameID INT NOT NULL UNIQUE,
        update_date DATETIME NOT NULL,
        description LONGTEXT,
        PRIMARY KEY (developerID, gameID, update_date),
        FOREIGN KEY (developerID) REFERENCES developer
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
        FOREIGN KEY (gameID) REFERENCES game
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

### 2.1.18 Develop

**Relational Model:**

develop(<u>developerID, gameID</u>)

**Functional Dependencies:**

None

**Candidate Key:**

{(developerID, gameID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE develop(
      developerID INT NOT NULL UNIQUE,
      gameID INT NOT NULL UNIQUE,
      PRIMARY KEY (developerID, gameID),
      FOREIGN KEY (developerID) REFERENCES developer
                          ON DELETE CASCADE
                          ON UPDATE CASCADE,
      FOREIGN KEY (gameID) REFERENCES game
                          ON DELETE CASCADE
                          ON UPDATE CASCADE
);
```

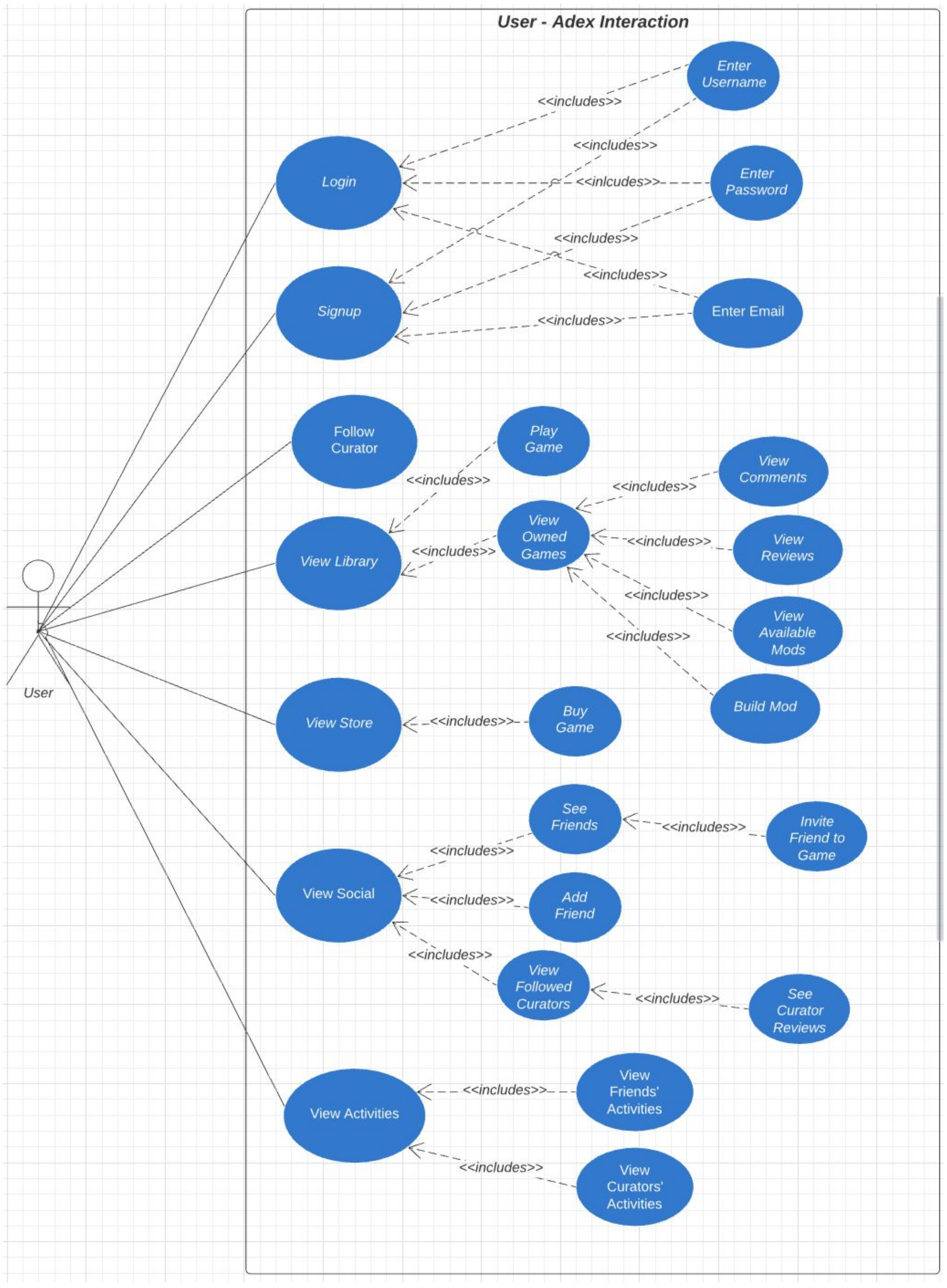
## 2.2 Functional Dependencies and Normalization of Tables

The functional dependencies and normalization of relational schemas are clearly indicated above for each relation. Since every schema is in BCNF form, there is no need for further normalization.

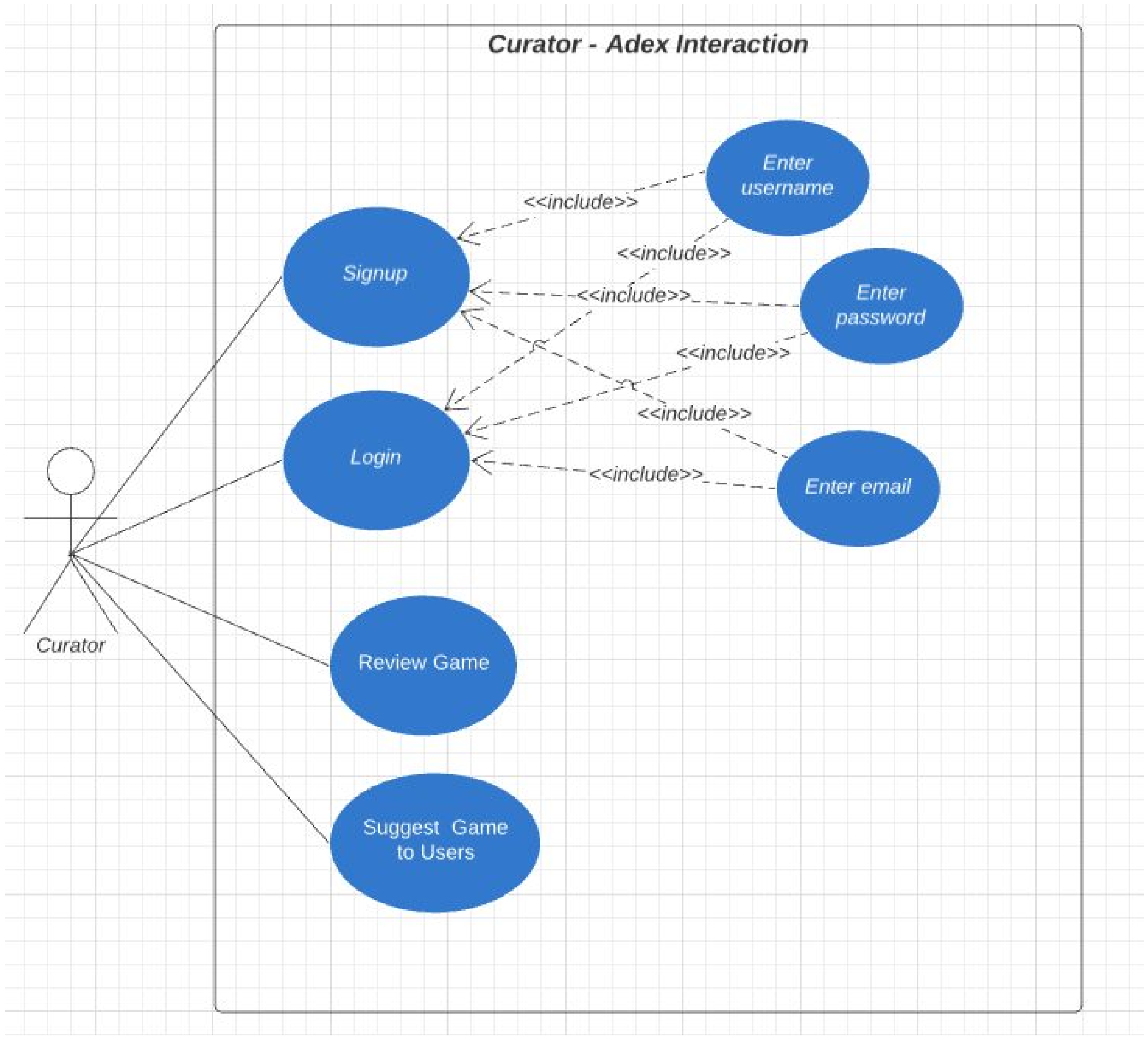
# 4 Functional Components

## 4.1 Use Case Model

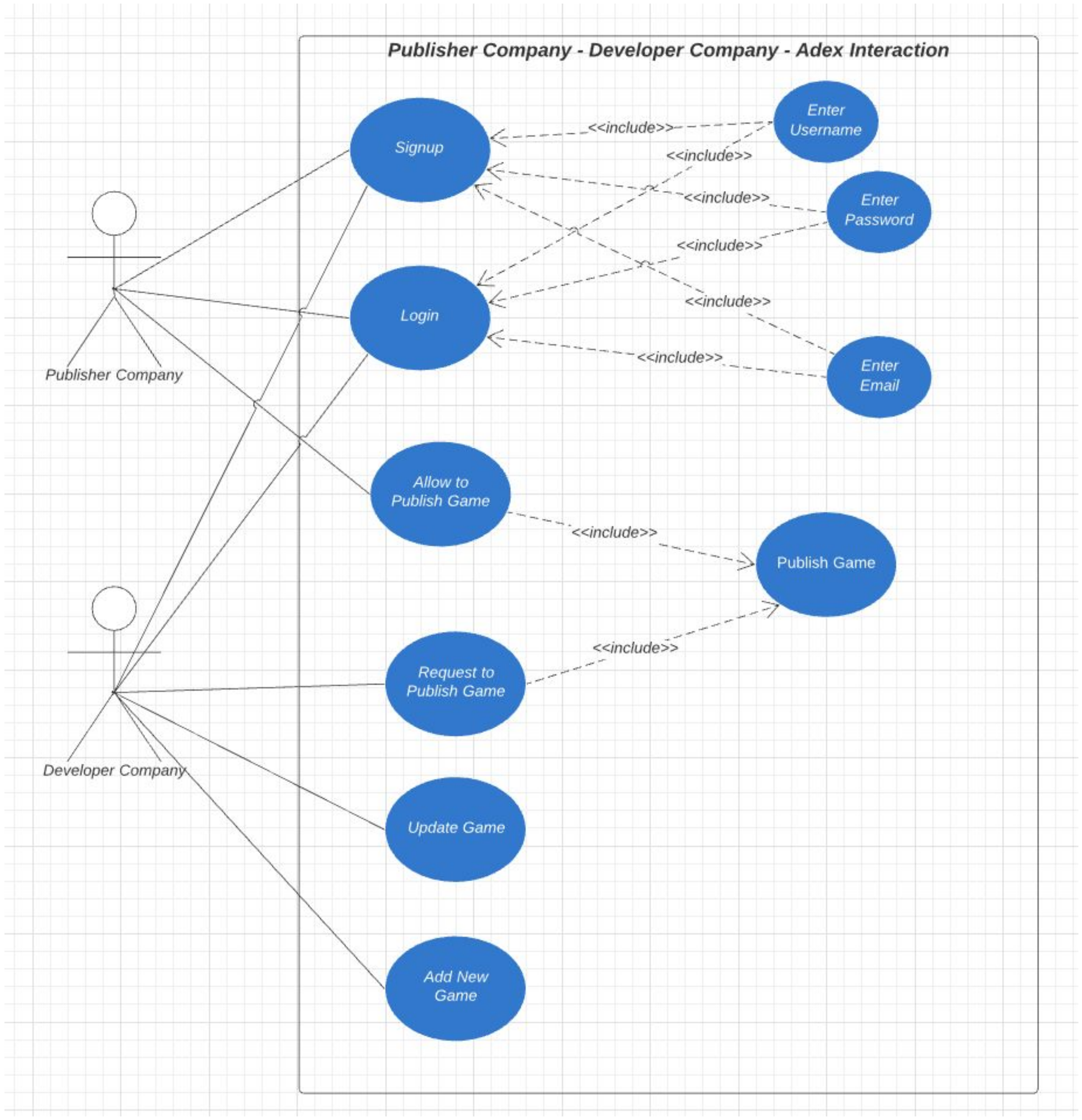
### 4.1.1 User Use Case Diagram



## 4.1.2 Curator Use Case Diagram



## 4.1.3 Publisher Company & Developer Company Use Case Diagram



## 4.2 Use Case Scenarios

### 4.2.1 Login Scenario

| ***Use Case Name:*** | **Login** |
|---|---|
| *Participating actors:* | User or Curator or Publisher Company or Developer Company (Referred to as ‘actor’ in this table) |
| *Stakeholders/Interests:* | Actor wants to login to the system with their credentials |
| *The flow of events:* | 1. Actor opens Adex website.<br><br>2. Actor enters their email or username to the related box.<br><br>3. Actor enters their password to the related box.<br><br>4. Actor clicks the ‘Sign In’ button.<br><br>5. Adex system is started for the actor. |
| *Pre-conditions:* | Actor should have signed up to Adex and have an account. |
| *Post-conditions:* | Actor is logged in to their Adex account. |
| *Exit:conditions:* | The entered username/email or password may be incorrect. |

### 4.2.2 Signup Scenario

| ***Use Case Name:*** | **Signup** |
|---|---|
| *Participating actors:* | User or Curator or Publisher Company or Developer Company (Referred to as ‘actor’ in this table) |
| *Stakeholders/Interests:* | Actor wants to sign up to the Adex service. |
| *The flow of events:* | 1. Actor opens Adex website. |

| | |
|---|---|
| | 2. Actor clicks the ‘Sign Up’ button.<br><br>3. Actor enters their username to the related box.<br><br>4. Actor enters their email to the related box.<br><br>5. Actor enters their password to the related box.<br><br>6. Actor clicks the ‘Sign Up’ button.<br><br>7. Adex system is set up and started for the actor. |
| *Pre-conditions:* | Actor should have an active email address to use for signup. |
| *Post-conditions:* | Actor has an Adex account created for them with the credentials they provided. |
| *Exit:conditions:* | None |

## 4.2.3 Follow Scenario

| ***Use Case Name:*** | **Follow Curator** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | Following a curator to see their activities on their social feed and being able to view their suggestions. |
| *The flow of events:* | 1. User opens the social tab on their account page.<br><br>2. User opens the curators section.<br><br>3. User selects a curator from the list and follows them.<br><br>4. The curator is added to the followed curators list of the user. |
| *Pre-conditions:* | The user must have an account on the Adex system, the user must be logged in the system, there must be at least one available curator not currently followed by the user. |

| | |
|---|---|
| *Post-conditions:* | The user is following the curator picked. |
| *Exit:conditions:* | The curator does not exist. |
| *Alternative Scenarios:* | 3.1 User searches for a curator to follow by their username. |

### 4.2.4 Play Scenario

| ***Use Case Name:*** | **Play Game** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | User plays one of their owned games. |
| *The flow of events:* | 1. User opens the library tab on their Adex account.<br><br>2. User selects one of the games in their library and clicks the ‘Play’ button next to it.<br><br>3. The game opens. |
| *Pre-conditions:* | User must have an Adex account, user must be logged in the system, user must have at least one bought game. |
| *Post-conditions:* | User plays the game. |
| *Exit:conditions:* | The game is not available for play. |

### 4.2.5 Build Game Scenario

| ***Use Case Name:*** | **Build Game Mod** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | User builds a mod to one of their owned games |

| | |
|---|---|
| *The flow of events:* | 1. User opens the library tab on their account.<br><br>2. User selects a game and clicks the ‘Build Mod’ button next to it.<br><br>3. User starts to build a mod for the selected game. |
| *Pre-conditions:* | User must have an Adex account, user must be logged in the system, user must have at least one bought game. |
| *Post-conditions:* | User has built a mode for their selected game. |
| *Exit:conditions:* | The game does not allow for mods to be built. |

### 4.2.6 Buy Game Scenario

| ***Use Case Name:*** | **Buy Game** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | User buys a game |
| *The flow of events:* | 1. User opens the store tab on their account.<br><br>2. User selects a game from the list of games.<br><br>3. User purchases the selected game. |
| *Pre-conditions:* | User must have an Adex account, user must be logged in the system |
| *Post-conditions:* | User has purchased the game they wanted. |
| *Exit:conditions:* | User’s balance is not enough to purchase the game. |

## 4.2.7 Invite Friend to Game Scenario

| ***Use Case Name:*** | **Invite Friend to Game** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | User invites a friend to a game to play together. |
| *The flow of events:* | 1. User opens the ‘Social’ tab on their account.<br>2. User opens the ‘Friends’ tab.<br>3. User selects a friend to invite to a game and clicks the ‘Invite’ button next to them.<br>4. User and their friend play together. |
| *Pre-conditions:* | User must have an Adex account, user must be logged in the system, user must have at least one purchased game, user must have at least one friend, user’s friend must own the same game. |
| *Post-conditions:* | User and their friend play a game together |
| *Exit:conditions:* | User’s friend is offline. |

## 4.2.8 Add Friend Scenario

| ***Use Case Name:*** | **Add Friend** |
|---|---|
| *Participating actors:* | User |
| *Stakeholders/Interests:* | User adds a friend to their friends list |
| *The flow of events:* | 1. User opens the ‘Social’ tab on their account.<br>2. User searches for the username of the friend they want to add via the search box.<br>3. User clicks the ‘Add friend’ button next to them. |

|  |  |
|---|---|
|  | 4. User and the searched user are now friends. |
| *Pre-conditions:* | User must have an Adex account, user must be logged in the system |
| *Post-conditions:* | User and the searched user are friends. |
| *Exit:conditions:* | The searched username does not match with any users on the Adex system. |

### 4.2.9 Review Game Scenario

| ***Use Case Name:*** | **Review Game** |
|---|---|
| *Participating actors:* | Curator |
| *Stakeholders/Interests:* | Curator reviews a game |
| *The flow of events:* | 1. Curator opens the ‘Store’ tab on their account.<br><br>2. Curator picks a game they want to review and clicks the ‘Add Review’ button next to it.<br><br>3. Curator writes their review to the related box.<br><br>4. Curator clicks the ‘Submit Review’ button. |
| *Pre-conditions:* | Curator must have an Adex account, curator must be logged in the system. |
| *Post-conditions:* | Curator has reviewed a game. |
| *Exit:conditions:* | There are no games in store to review. |

### 4.2.10 Suggest Game Scenario

| ***Use Case Name:*** | **Suggest Game** |
|---|---|

| | |
|---|---|
| *Participating actors:* | Curator |
| *Stakeholders/Interests:* | Curator suggests a game to users |
| *The flow of events:* | 1. Curator opens the ‘Store’ tab on their account.<br><br>2. Curator selects a game to suggest, and clicks the ‘Suggest Game’ button next to it. |
| *Pre-conditions:* | Curator must have an Adex account, curator must be logged in the system. |
| *Post-conditions:* | Curator has suggested a game to their followers. |
| *Exit:conditions:* | There are no games in store to suggest. |

## 4.2.11 Publish Game Scenario

| ***Use Case Name:*** | **Publish Game** |
|---|---|
| *Participating actors:* | Publisher Company, Developer Company |
| *Stakeholders/Interests:* | Developer Company requests the Publisher Company to publish a game, Publisher company accepts the request and publishes the game. |
| *The flow of events:* | 1. Developer Company requests a Publisher Company to publish a game they developed.<br><br>2. Publisher Company accepts the request.<br><br>3. Publisher Company publishes the game. |
| *Pre-conditions:* | Publisher Company must have an Adex account, Developer Company must have an Adex account, Publisher Company must be logged in to the system, Developer Company must be logged in to the system |
| *Post-conditions:* | The game is published in the Adex store. |

| | |
|---|---|
| *Exit:conditions:* | Publisher Company declines Developer Company's request. |

### 4.2.12 Add New Game Scenario

| ***Use Case Name:*** | **Add New Game** |
|---|---|
| *Participating actors:* | Developer Company |
| *Stakeholders/Interests:* | Developer Company develops and adds a new game to their account (they may later request a Publisher Company to publish it in the Adex Store). |
| *The flow of events:* | 1. Developer company develops a game.<br><br>2. Developer company adds the developed game to their account on Adex by clicking the 'Add New Game' button on their account. |
| *Pre-conditions:* | Developer Company must have an Adex account, Developer Company must be logged in the system |
| *Post-conditions:* | Developer Company has added a new game to the system. |
| *Exit:conditions:* | None |

## 4.3 Algorithms

### 4.3.1 Most Liked Games

This algorithm obtains K number of games which are most liked games by users and the list should be in descending order. First, the comment table should be grouped by gameIDs. This table includes every rating given by any user. In order to calculate the average rating of any game, the summation of ratings for each game will be divided by the number of ratings (or comments). Lastly, each game should be sorted by its average rating and top K number of games should be returned.

### 4.3.2 Most Recommended Games By Curators

This algorithm is similar to the "Most Liked Games" algorithm, the algorithm retrieves the top K games that have the highest average rating from the curators. The grouping by gameIDs need to be done on review table. Then, the average of ratings should be calculated for each game. After sorting, top K games with the highest average ratings should be returned.

### 4.3.3 Viewing Friends' Activities

Viewing friends' activities needs an algorithm which uses multiple relation tables. Firstly, the friends of the user are obtained from friend relation. Then, new friendships, comments, new game purchases, mod creations of the friends will be retrieved from friend, comment, library, build relations, respectively. Dates of activities should be checked and only the activities occurred in the last week will be returned.

### 4.3.4 Logical Requirements

Since different relations have references to each other, this may cause some logical errors. In order to prevent those kind of problems, referential integrity of relations is defined on the table creation. Other possible error prone events will be handled in the algorithm design process. For example, user will not be able to send friend request to his/her friends and this condition will be checked from friend and friend_request table.

## 4.4 Data Types

In the implementation of the project and in the definition of relation tables, numeric type, String type and Data-Time type are being used for the attribute domains.

Numeric type data types **INT**, **DECIMAL** and **BOOLEAN** are used to store different values such as balance, id of users, discount.

**VARCHAR** and **LONGTEXT** are used to define the domain of texts, descriptions, emails and the other string-like attributes.

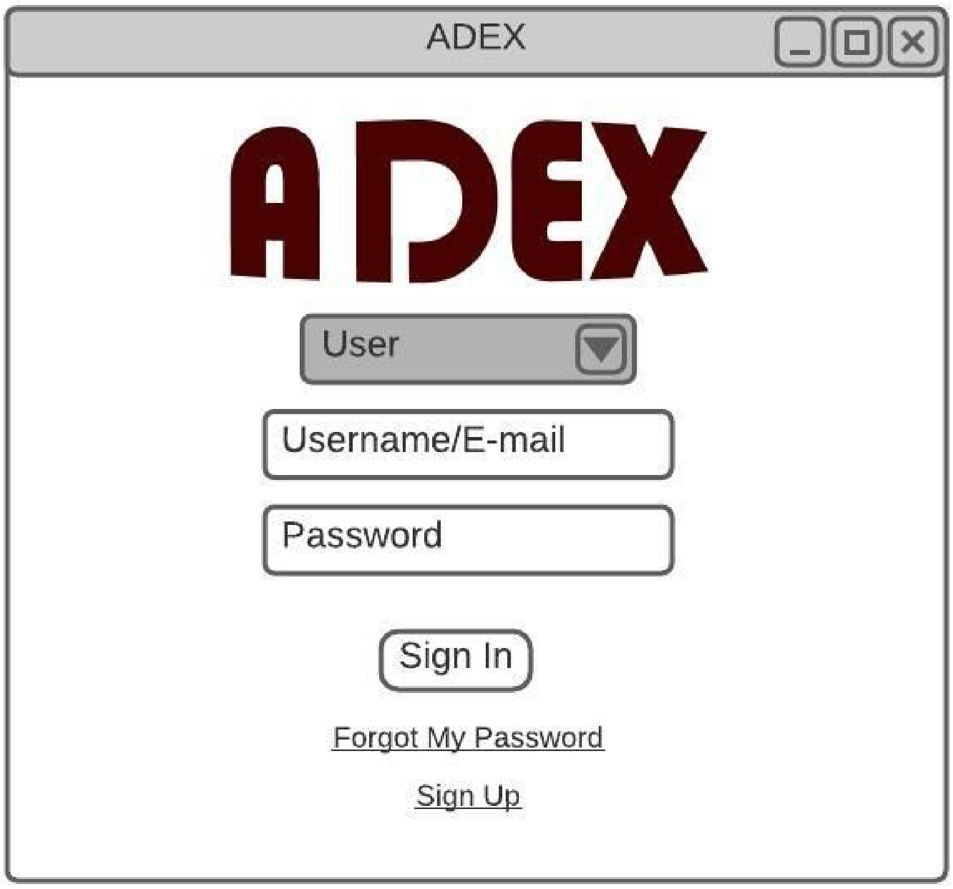
Lastly, **DATETIME** is used as the data type of dates of necessary events.

# 5 User Interface Design & Corresponding SQL Statements

In this section, we show the user interface design and SQL statements of our project. Users, curators, developer companies and publisher companies will give a different user interface.

## 5.1 Users

### 5.1.1 Sign In Page



**Input:** @email_username, @password, @user_name, @email

**Process:** When the users open ADEX, they will be welcomed with our login page. If the users are not already signed up, they can go to the sign-up page after clicking the “Sign Up” link. Also, in the case of forgetting a password, users can change it through the “Forgot My Password” link. After logging in, the users will be directed to their library.

## SQL Statements:

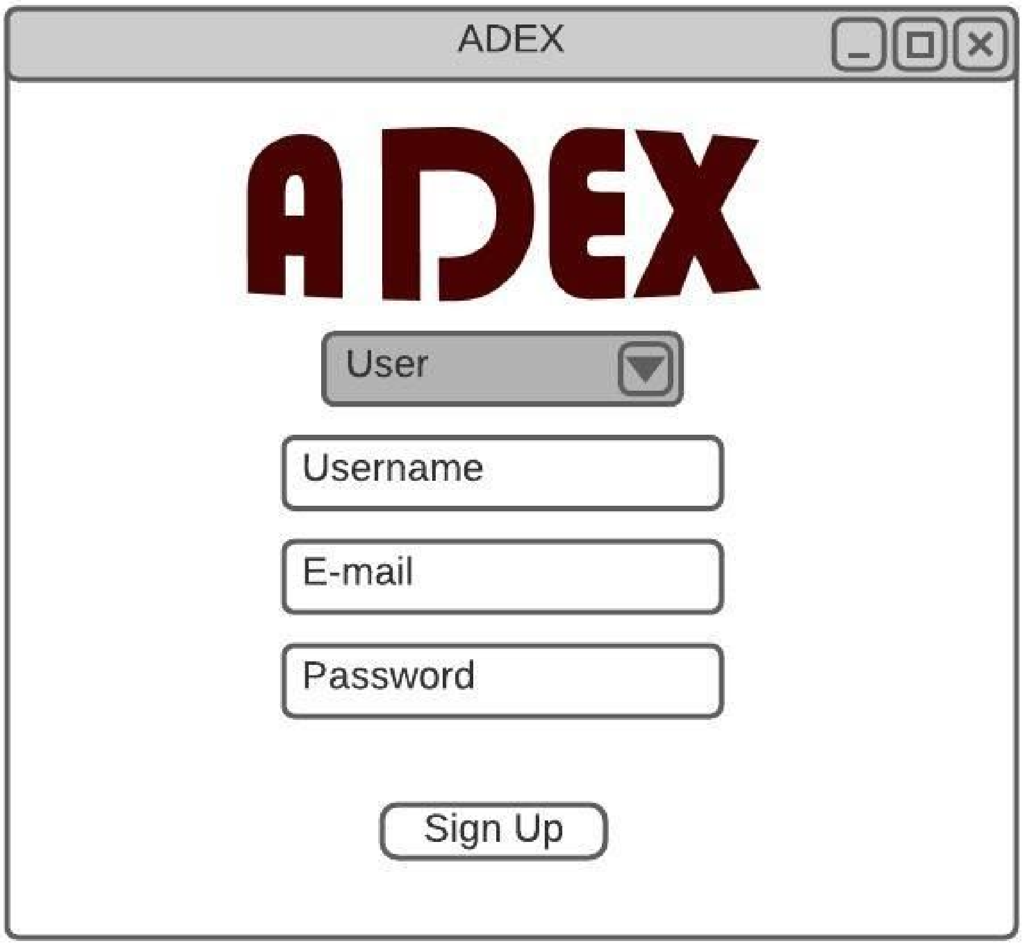
### Log - in:

```
/*
Inputs: @email_username, @password
*/

SELECT userID
FROM User
WHERE (email = @email_username OR username = @email_username) AND password =
@password
```

### Changing Password:

```
/*
Inputs: @userID, @password
*/

UPDATE User
SET password = @password
WHERE userID = @userID
```

## 5.1.2 Sign Up Page



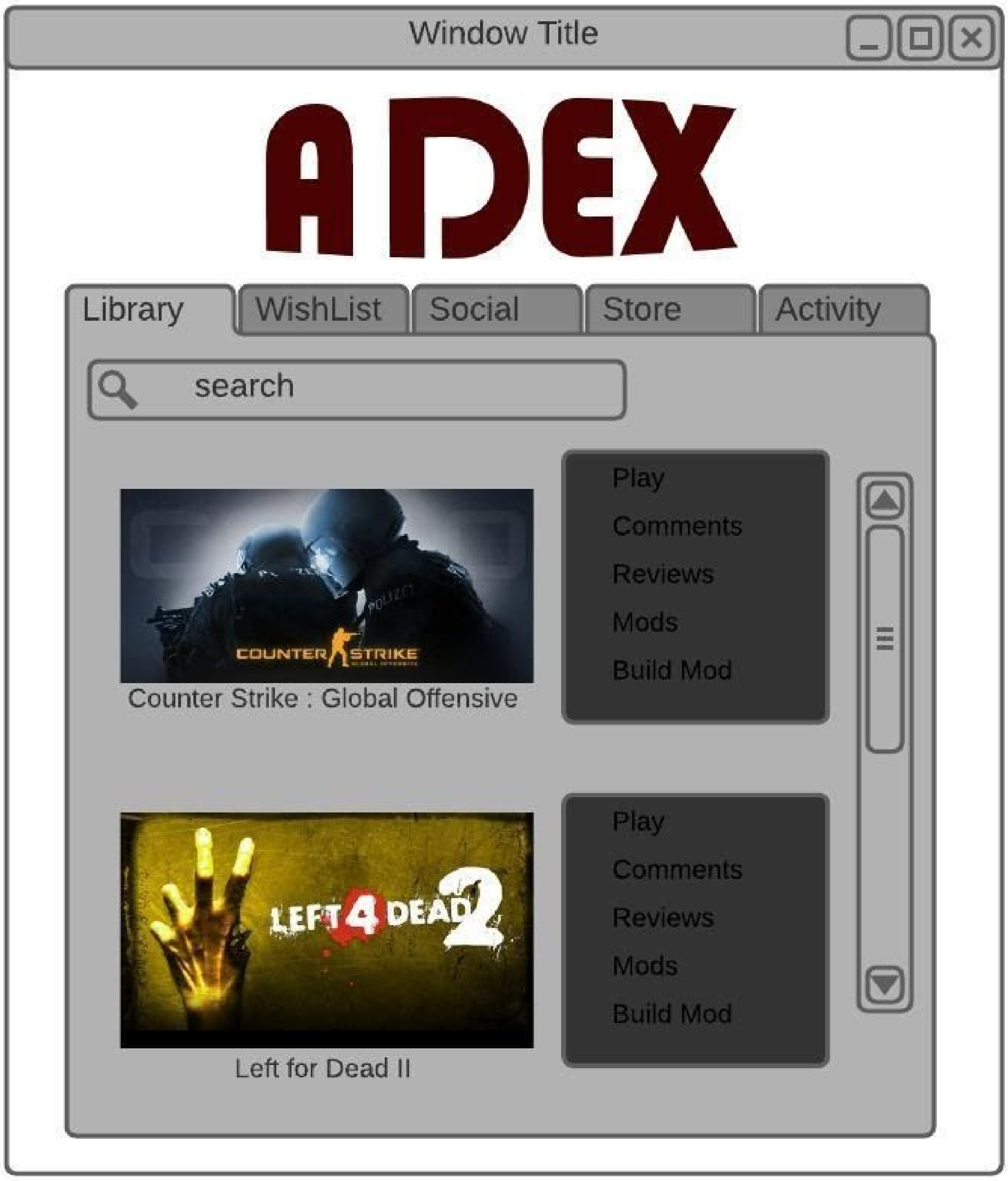
**Input:** @username, @email, @password

**Process:** When the users want to sign in, they will be directed to this page. After signing up, they will be directed to login page

**Sign up:**

```
/*
Inputs: @username, @email, @password
*/

INSERT INTO User (username, password, email) VALUES (
        @username,
        @password,
        @email
);
```

## 5.1.3 Library



**Input:** @userID

**Process:** In the library section, users can see the games they have. Users can take different actions for a game. If they want to play the game, they can click the “Play” button and start playing. If they want to make a comment or see the comments made by other users, they can click the “See Comments” button. If they want to see the reviews made by curators, they can click the “See Reviews” button. If they want to play the game in a different mode created by some user they can click the “See

Mods” button and select the mode they want to play. In addition, if they want to build a new mod they can click the “Build Mod” button and start building.

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @userID
*/

SELECT G.game_name, G.image
FROM Library L natural join Game M natural join User U
WHERE U.userID = @userID
ORDER BY game_name
```

**See Comments:**

```
/*
Inputs: @gameID
*/

SELECT C.text, C.rating, G.game_name
FROM Comment C natural join Game G
WHERE gameID = @gameID
```

**See Review:**

```
/*
Inputs: @gameID
*/

SELECT R.text, R.rating, G.game_name, C.username
FROM Review R natural join Game G natural join Curator C
WHERE gameID = @gameID
```

**See Mods:**

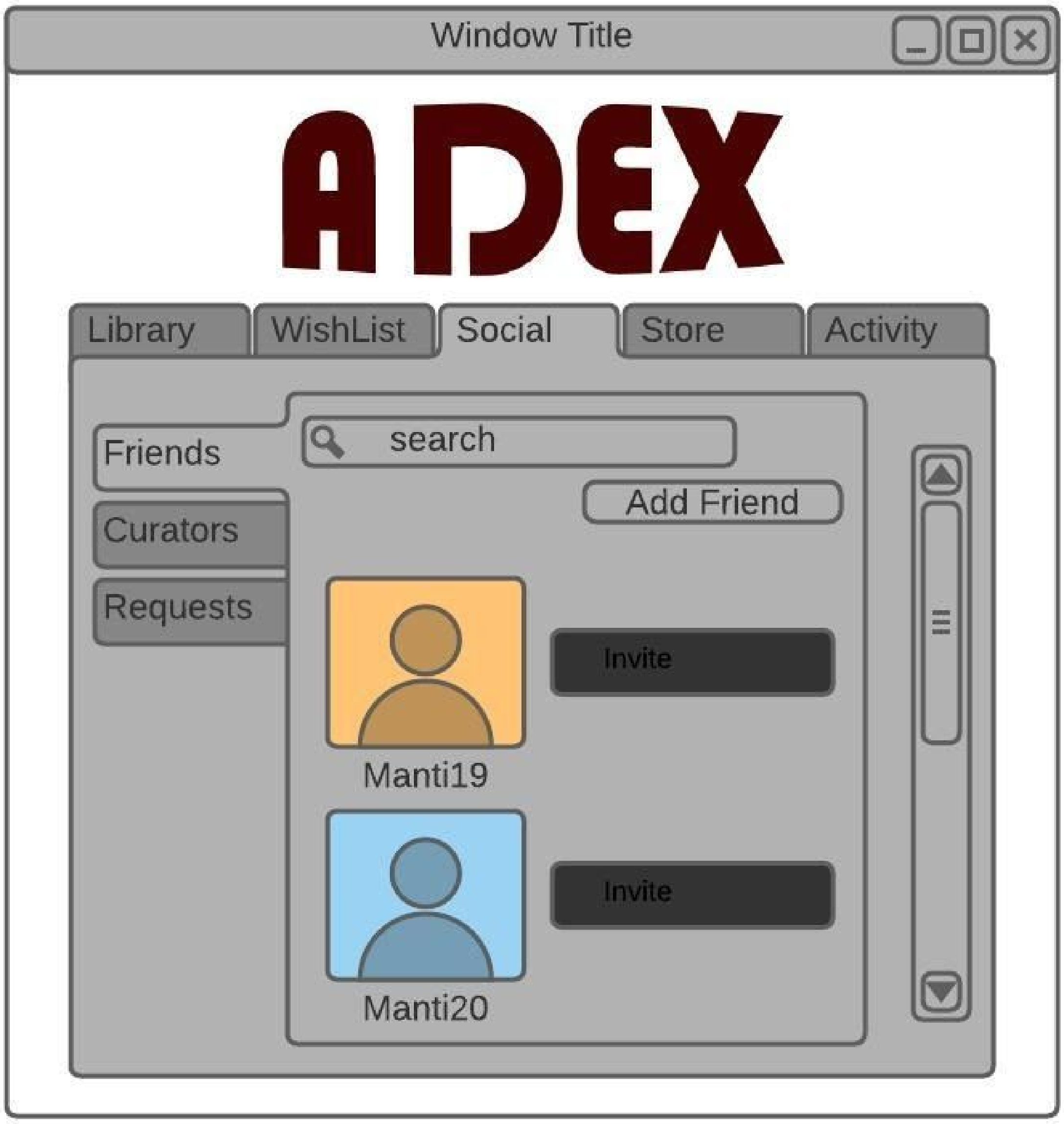
```
/*
Inputs: @gameID
*/

SELECT B.mod
FROM Build B natural join Game G
WHERE gameID = @gameID
```

**Build Mod:**

```
/*
Inputs: @userID, @gameID, @mod
*/

INSERT INTO Build(gameID, userID, mod) VALUES (
	@gameID,
	@userID,
	@mod
);
```

## 5.1.4 Social

### 5.1.4.1 Friends



**Input:** @userID

**Process:** In the Social section, users can see their friends and invite them to a game they own by clicking the “Invite” button. Also, Users can add new friends.

## SQL Statements:

### Viewing Friends:

```
/*
Inputs: @userID
*/

SELECT U1.username, U1.image
FROM Friend F, User U1, User U2
WHERE U2.userID = @userID AND U1.userID <> @userID AND ((F.userID2 =
U1.userID AND F.userID1 = U2.userID) OR (F.userID1 =  U1.userID AND F.userID2 =
U2.userID))
ORDER BY U1.username
```

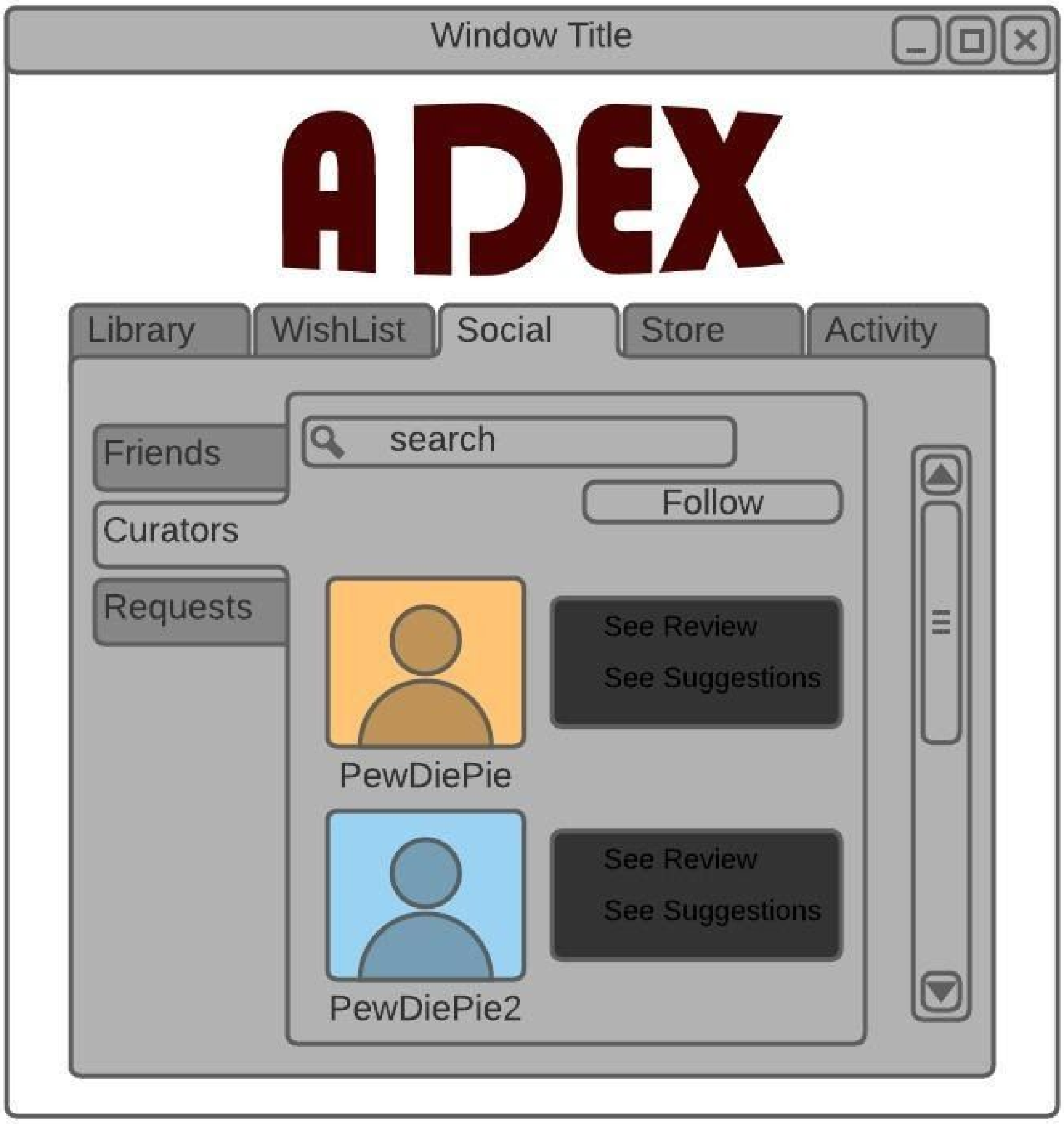
### Inviting Friends to games:

```
/*
Inputs: @userID, @gameID, @userID2, @date
*/

INSERT INTO Invitation(userID1, userID2, gameID, date) VALUES (
        @userID,
        @userID2,
        @gameID,
        @date
);
```

### Sending Friend Request:

```
/*
Inputs: @userID, @userID2
*/

INSERT INTO Friend_Request(userID1, userID2) VALUES (
        @userID,
        @userID2
);
```

5.1.4.2 Curators



**Input:** @userID

**Process:** In the Social section, users can see the curators they follow, see their reviews and see the suggested game by the curator. Also, Users can follow new curators.

## SQL Statements:

### Viewing Curators:

```
/*
Inputs: @userID
*/

SELECT C.username, C.image
FROM Curator C natural join Follow F natural join User U
WHERE U.userID = @userID
ORDER BY C.username
```

### Following New Curators:

```
/*

Inputs: @userID, @curatorID
*/

INSERT INTO Follow(userID, curatorID) VALUES (
        @userID,
        @curatorID
);
```

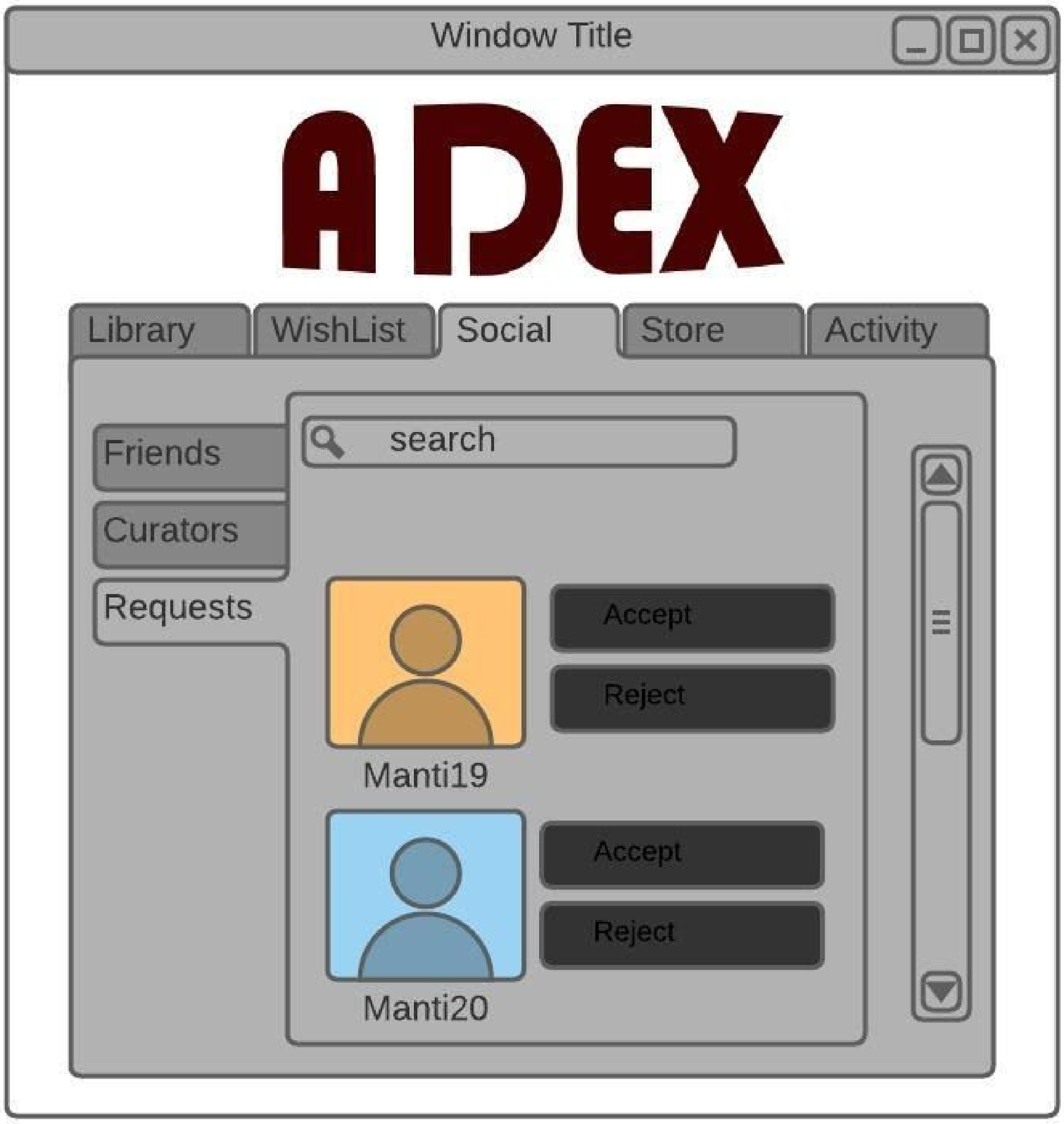
### Seeing Reviews of Curators:

```
/*
Inputs: @curatorID, @userID
*/

SELECT R.text, R.rating, G.game_name
FROM Review R natural join Curator C natural join Follow F natural join User U natural
join Game G
WHERE C.curatorID= @curatorID AND U.userID = @userID
```

**Seeing Suggestions by Curators:**

```
/*
Inputs: @curatorID,, @userID
*/

SELECT G.game_name
FROM User U natural join Curator C natural join Follow F natural join Game G
WHERE C.curatorID= @curatorID AND U.userID = @userID
```

### 5.1.4.3 Requests



**Input:** @userID

**Process:** In the Social section, users can see the friend requests they get. If they accept the request they will have a one new friend.

**SQL Statements:**

**Viewing Friend Requests:**

```
/*
Inputs: @userID
*/

SELECT U1.username, U1.image
FROM Friend_Request F, User U1, User U2
WHERE U2.userID = @userID AND U1.userID <> @userID AND F.userID2 =  U1.userID
AND F.userID1 = U2.userID
ORDER BY U1.username
```

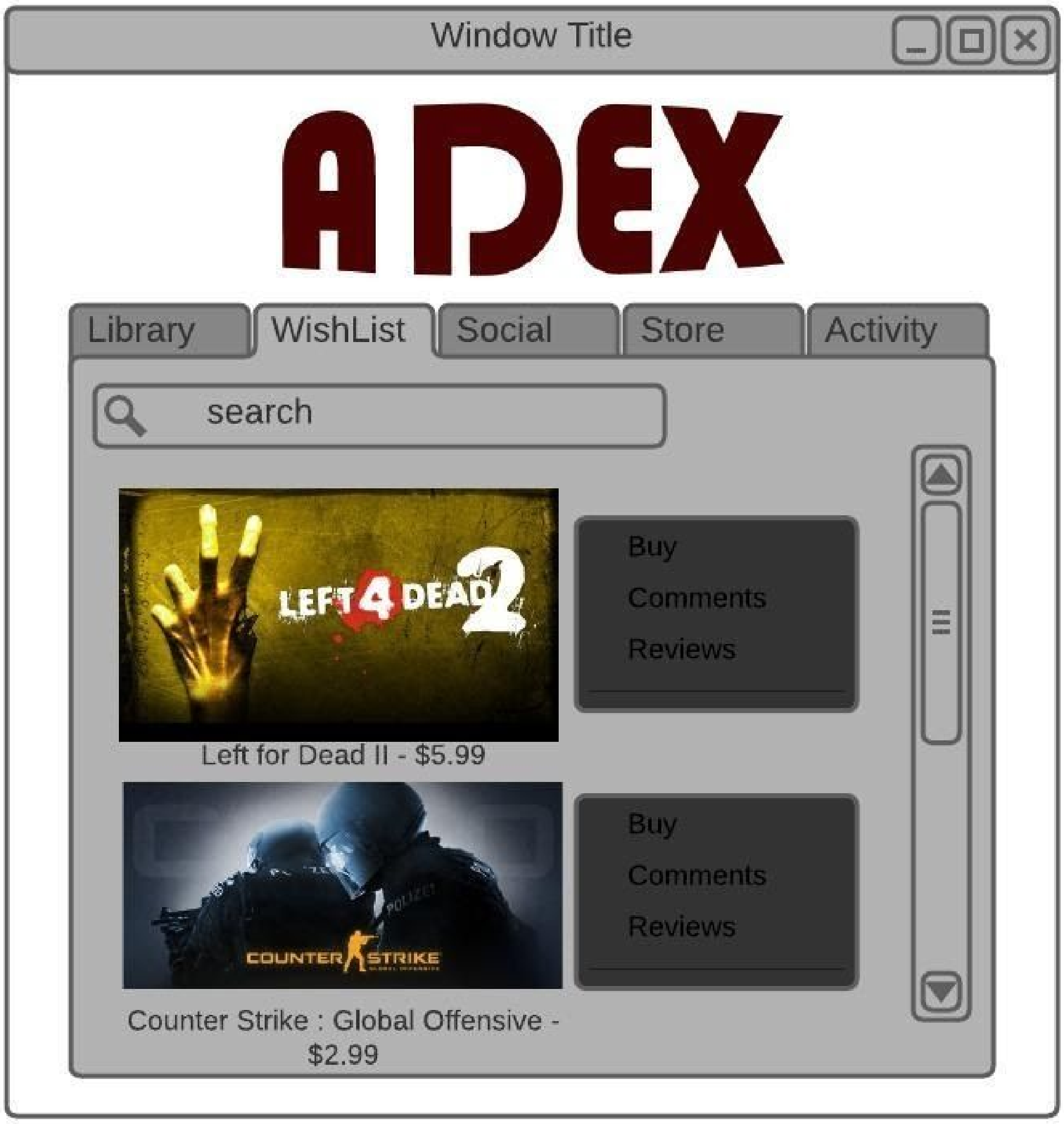
**Accepting the Request**

```
/*
Inputs: @userID, @userID2, @date
*/

INSERT INTO Friend(userID1, userID2, date) VALUES (
        @userID,
        @userID2
        @date
);

DELETE FROM Friend_Request(userID1, userID2)
WHERE userID1 = @userID2 AND userID2 = @userID
```

**Rejecting the Request**

```
/*
Inputs: @userID, @userID2
*/

DELETE FROM Friend_Request(userID1, userID2)
WHERE userID1 = @userID AND userID2 = @userID2
```

## 5.1.5 Wishlist



**Input:** @userID

**Process:** In the wishlist section, users can see the games they have added to their wishlist. Users can take different actions for a game. If they want to buy the game, they can click the “Buy” button and buy the game. If they want to make a comment or see the comments made by other users, they can click the “Comments” button. If they want to see the reviews made by curators, they can click the “Reviews” button.

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @userID
*/

SELECT G.game_name, G.image, G.price
FROM Wishlist W natural join Game M natural join User U
WHERE U.userID = @userID
ORDER BY game_name
```

**Buying a Game:**

```
/*
Inputs: @userID, @date, @gameID
*/

INSERT INTO Library(userID, gameID, date) VALUES (
        @userID,
        @gameID
        @date
);

DELETE FROM Wishlist(userID, gameID)
WHERE userID = @userID AND gameID = @gameID
```

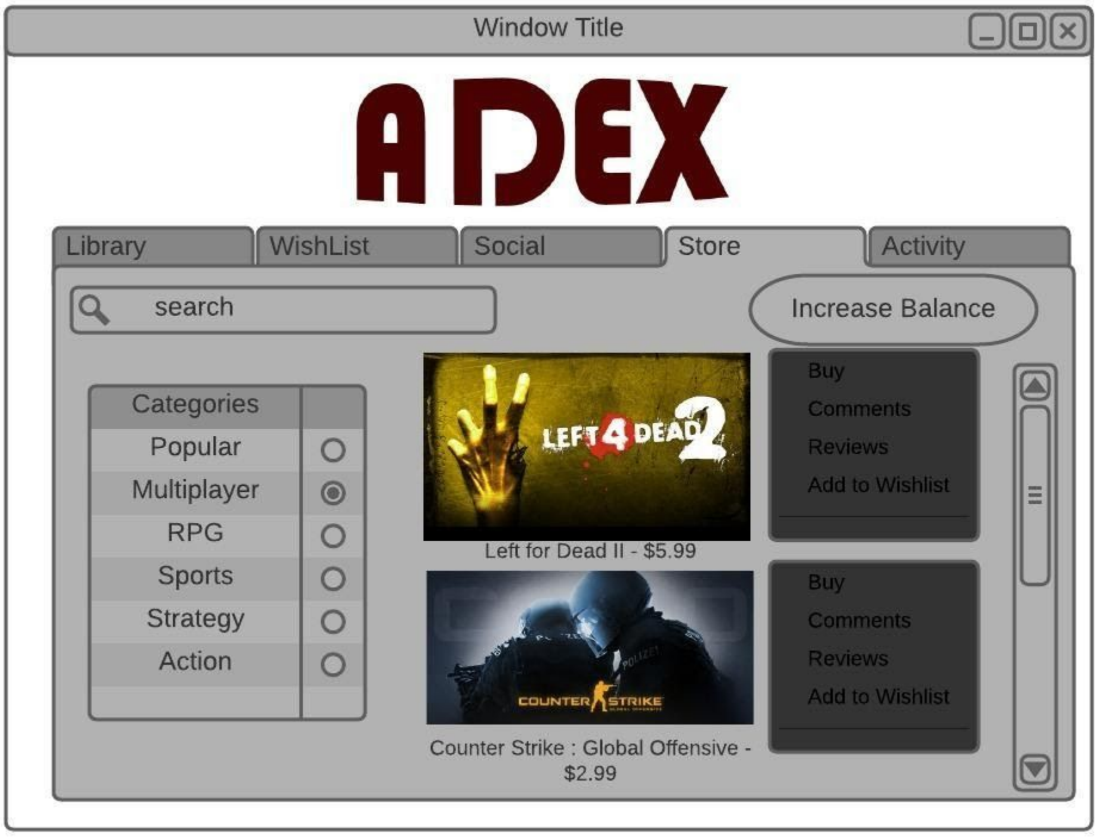
**See Comments:**

```
/*
Inputs: @gameID
*/

SELECT C.text, C.rating, G.game_name
FROM Comment C natural join Game G
WHERE gameID = @gameID
```

**See Reviews:**

```
/*
Inputs: @gameID
*/

SELECT R.text, R.rating, C.username
FROM Review R natural join Game G natural join Curator C
WHERE gameID = @gameID
```

### 5.1.5 Store



**Input:** @userID

**Process:** In the store section, users can see the games they haven’t bought yet on the platform. Users can increase their balance by clicking the “Increase Balance” button. Users can take different actions for a game. If they want to buy the game, they can click the “Buy” button and buy the game. If they want to make a comment or see the comments made by other users, they can click the “Comments” button. If they want to see the reviews made by curators, they can click the “Reviews” button.

Also, if they want to add the game to their wishlist, they can click the “Add to Wishlist” button.

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @userID, @genre
*/

SELECT G.game_name, G.image, G.price
FROM (SELECT  *
        FROM Game G, User U, Library L
        WHERE G.genre = genre AND G.active = “Yes”
        EXCEPT
        (SELECT *
        FROM Game G natural join User U natural join Library L)
        )
ORDER BY game_name
```

**Buying a Game:**

```
/*
Inputs: @userID, @date, @gameID
*/

INSERT INTO Library(userID, gameID, date) VALUES (
        @userID,
        @gameID
        @date
);
```

**See Comments:**

```
/*
Inputs: @gameID
*/

SELECT C.text, C.rating, G,game_name, U.username
FROM Comment C natural join Game G natural join User U
WHERE gameID = @gameID
```

**See Reviews:**

```
/*
Inputs: @gameID
*/

SELECT R.text, R.rating, C.username
FROM Review R natural join Game G natural join Curator C
WHERE gameID = @gameID
```

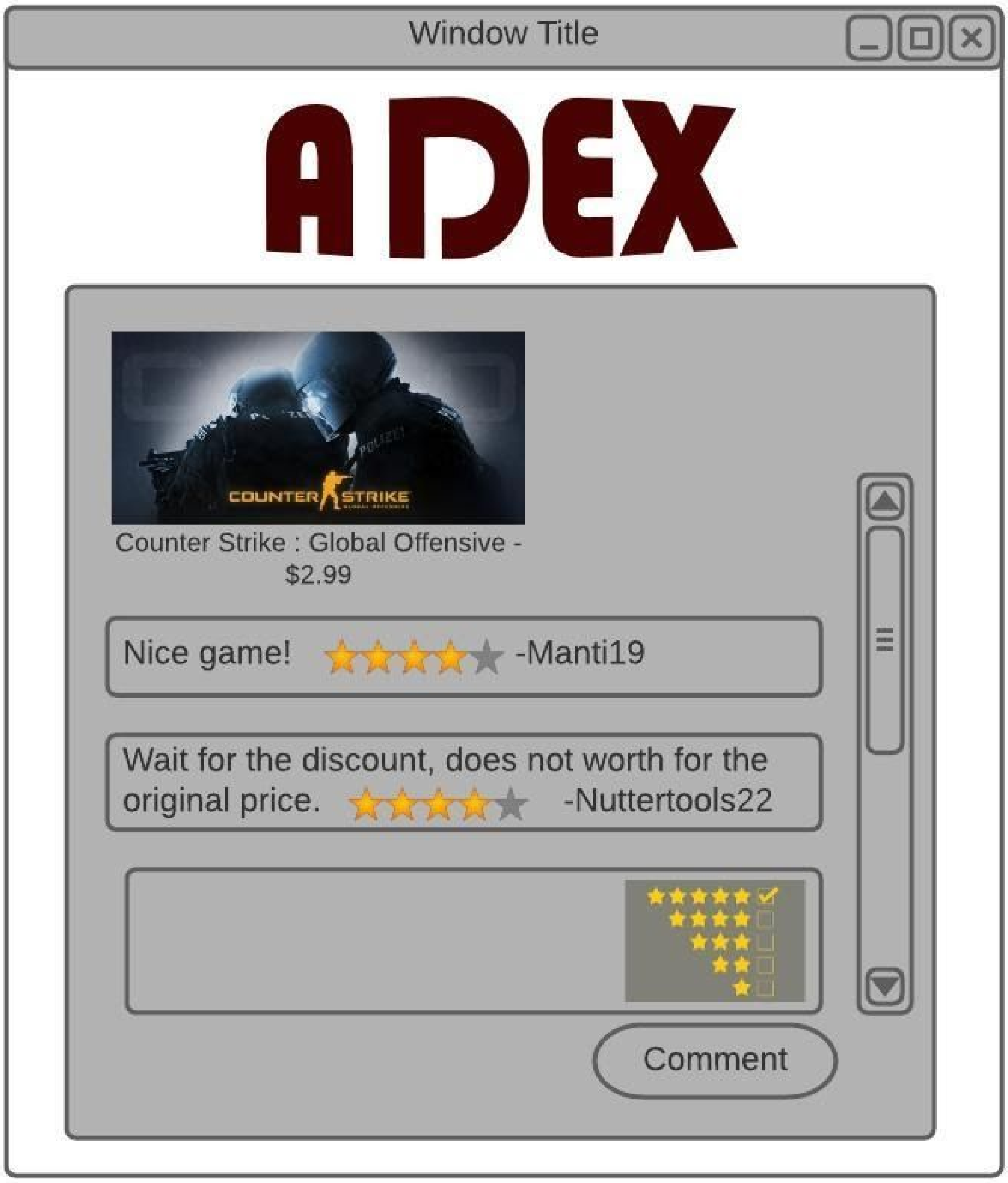
**Add To Wishlist:**

```
/*
Inputs: @userID, @gameID
*/

INSERT INTO Wishlist(userID, gameID) VALUES (
        @userID,
        @gameID
);
```

**Increase Balance:**

```
/*
Inputs: @userID, @amount
*/

UPDATE User
SET balance = balance + @amount
WHERE userID = @userID
```

## 5.1.6 Comments



Nice game! -Manti19

Wait for the discount, does not worth for the original price. -Nuttertools22

**Input:** @userID, @GameID

**Process:** When users click the “Comments” button either on their Library or in the Store section, they will be directed to this page.

**SQL Statements:**

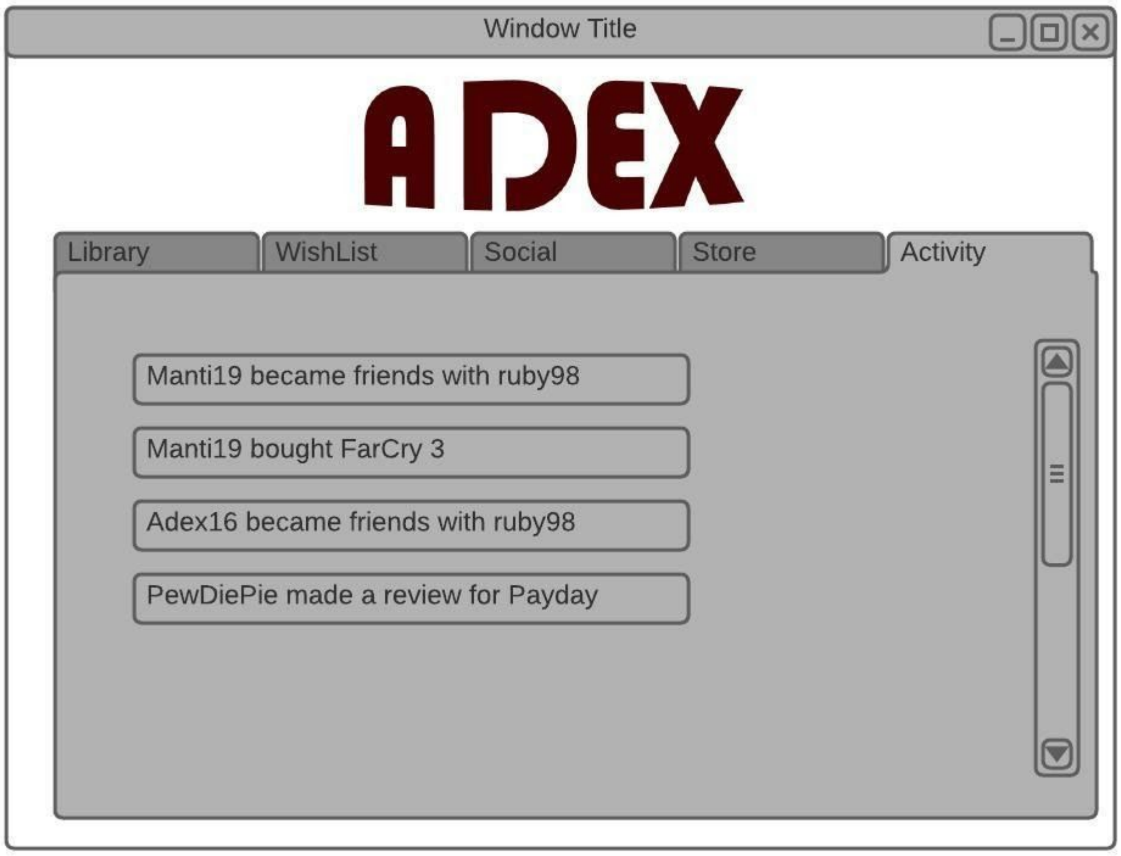
**See Comments:**

```
/*
Inputs: @gameID
*/

SELECT C.text, C.rating, U.username
FROM Comment C natural join Game G natural join User U
WHERE gameID = @gameID
```

**Make Comment:**

```
/*
Inputs: @userID, @gameID, @text, @rating
*/

INSERT INTO Comment(userID, gameID, text, rating) VALUES (
        @userID,
        @gameID,
        @text,
        @rating
);
```

## 5.1.7 Activity



Manti19 became friends with ruby98

Manti19 bought FarCry 3

Adex16 became friends with ruby98

PewDiePie made a review for Payday

**Input:** @userID

**Process:** In the Activity section, the user will be able to see the activities of their friends and followed curators

**SQL Statements:**

**Viewing Add Friend Activities of Friends :**

```
/*
Inputs: @userID
*/

WITH Friends AS ((SELECT userID2 AS friendID
                  FROM Friend
                  WHERE userID1 = @userID)
                  UNION
                  (SELECT userID1 AS friendID
                  FROM Friend
                  WHERE userID2 = @userID))
```

```
SELECT userID1 as friend1, userID2 as 2_deg_friend, F.date
FROM Friend F, Friends FF
WHERE (F.userID1 = FF.friendID OR F.userID2 = FF.friendID) AND F.date >=
(DATE(NOW()) - INTERVAL 5 DAY)
```

**Viewing Buy Game Activities of Friends :**

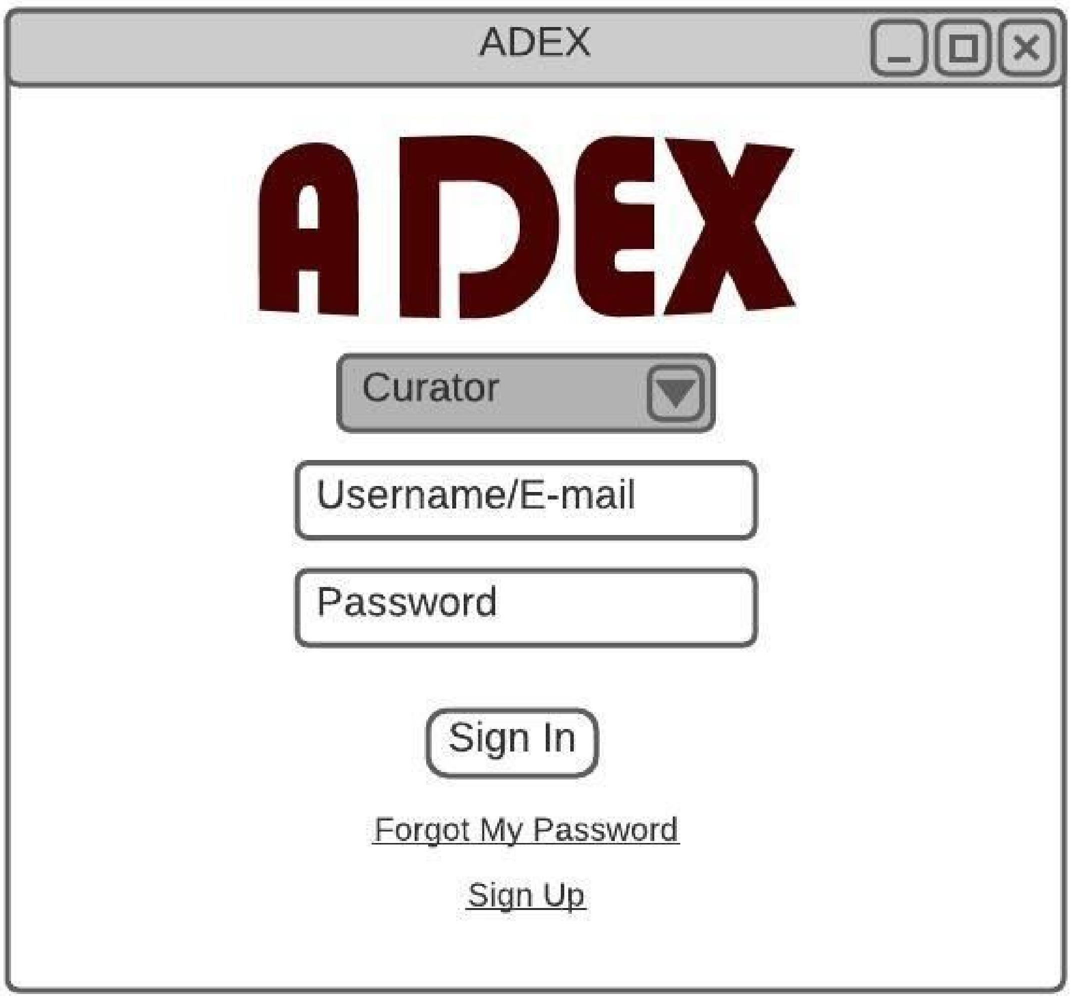
```
/*
Inputs: @userID
*/

WITH Friends AS ((SELECT userID2 AS friendID
                  FROM Friend
                  WHERE userID1 = @userID)
                  UNION
                  (SELECT userID1 AS friendID
                  FROM Friend
                  WHERE userID2 = @userID))
SELECT G.gameID, F.friendID, L.date
FROM Friends FF natural join Library L
WHERE L.date >= (DATE(NOW()) - INTERVAL 5 DAY)
```

**Viewing Activities of Curators:**

```
/*
Inputs: @userID
*/

SELECT C.username, G.game_name
FROM Curator C natural join Review R natural join Game G natural join Follow F natural
join User U
WHERE U.userID =  @userID

SELECT C.username, G.game_name
FROM Curator C natural join Suggest S natural join Game G natural join Follow F natural
join User U
WHERE U.userID =  @userID
```

## 5.2 Curators

### 5.2.1 Sign In Page



**Input:** @email_username, @password, @user_name, @email

**Process:** When the curators open ADEX, they will be welcomed with our login page. If the curators are not already signed up, they can sign up and go to the sign-up page after clicking the “Sign Up” link. Also, in the case of forgetting a password, curators can change it through the “Forgot My Password” link. After logging in, the curators will be directed to the store.

## SQL Statements:

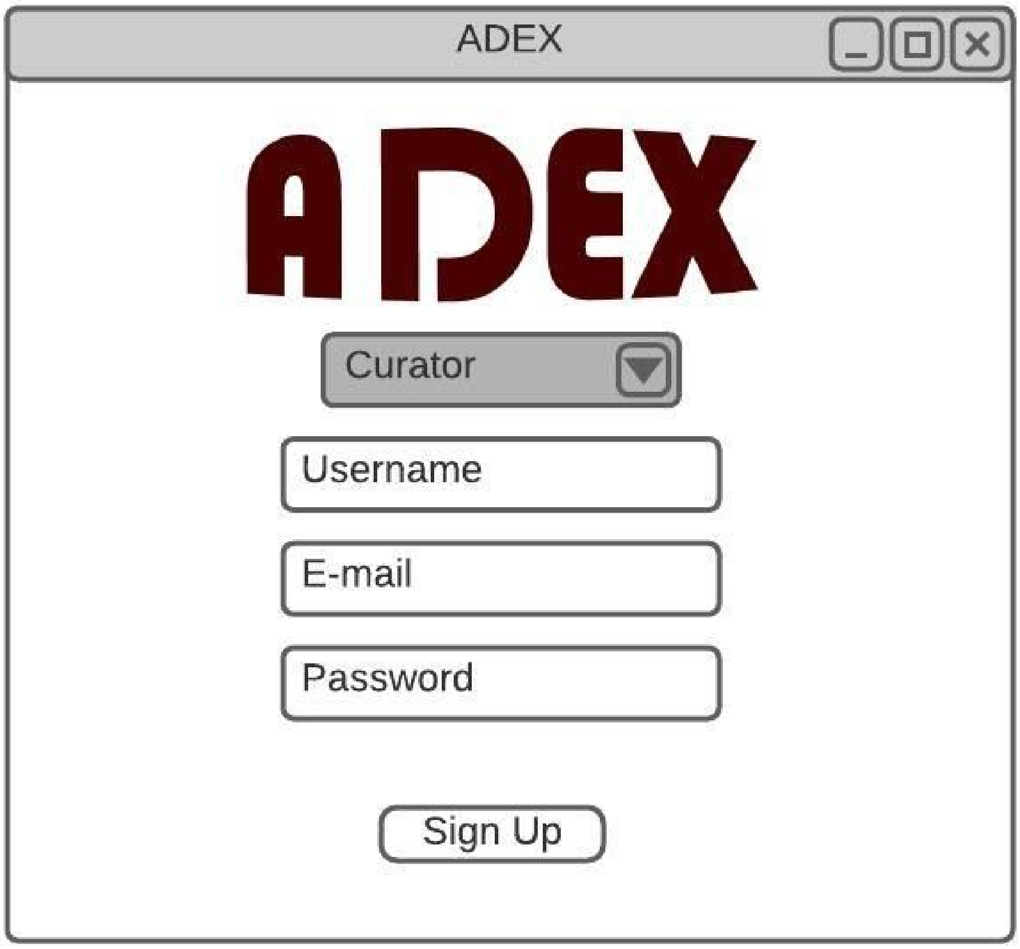
### Log - in:

```
/*
Inputs: @email_username, @password
*/

SELECT curatorID
FROM Curator
WHERE email = @email_username OR username = @email_username AND password =
@password
```

### Changing Password:

```
/*
Inputs: @curatorID, @password
*/

UPDATE Curator
SET password = @password
WHERE curatorID = @curatorID
```

## 5.2.2 Sign Up Page



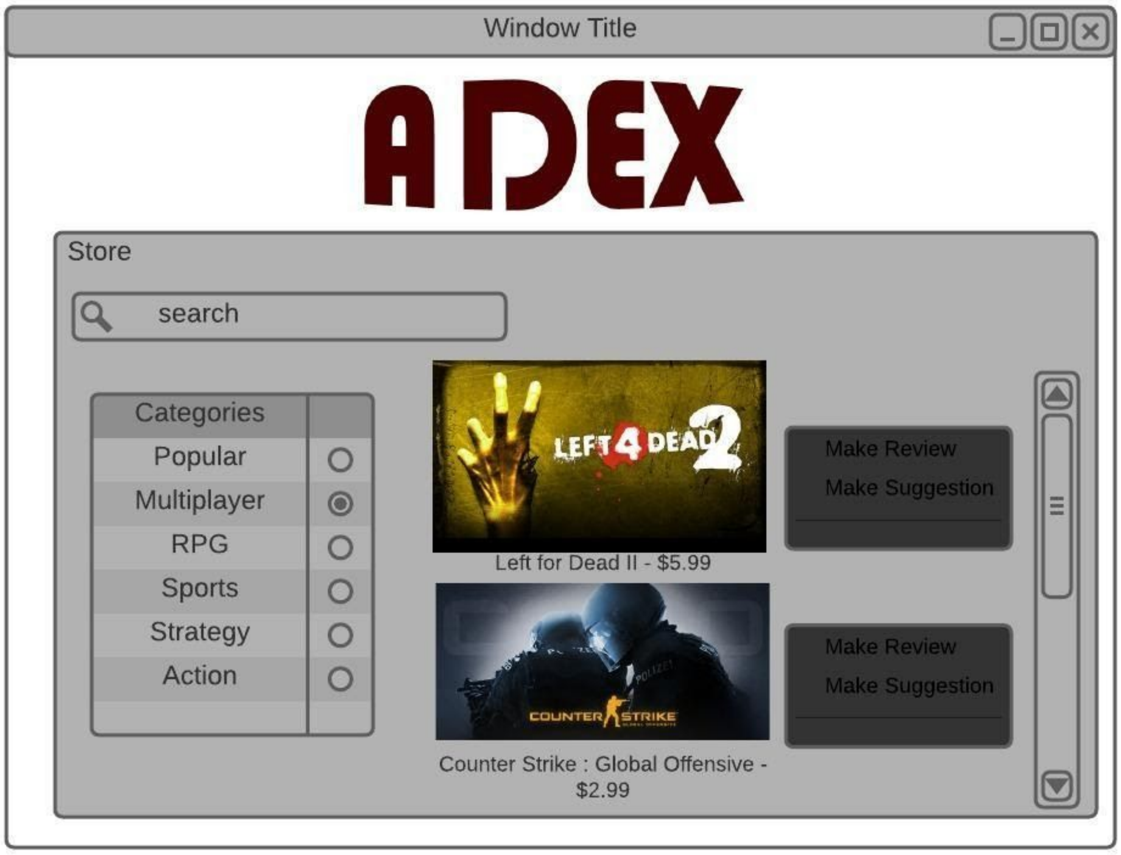
**Input:** @username, @email, @password

**Process:** When the curators want to sign up, they will be directed to this page. After signing up, they will be directed to the login page.

**Sign up:**

```
/*
Inputs: @username, @email, @password
*/

INSERT INTO Curator(username, password, email) VALUES (
        @username,
        @password,
        @email
);
```

## 5.2.3 Store



**Input:** @curatorID

**Process:** In the store section, curators can see the games on the platform. If they want to make a new review, they can click on the "Make Review" button

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @curatorID, @genre
*/


SELECT game_name, image, price
FROM Game
WHERE genre = @genre
```

### Making a new Review:

```
/*
Inputs: @curatorID, @gameID, text
*/

INSERT INTO Review(curatorID, gameID, text) VALUES (
        @curatorID,
        @gameID,
        @text
);
```

### List Followers to Suggest a Game:

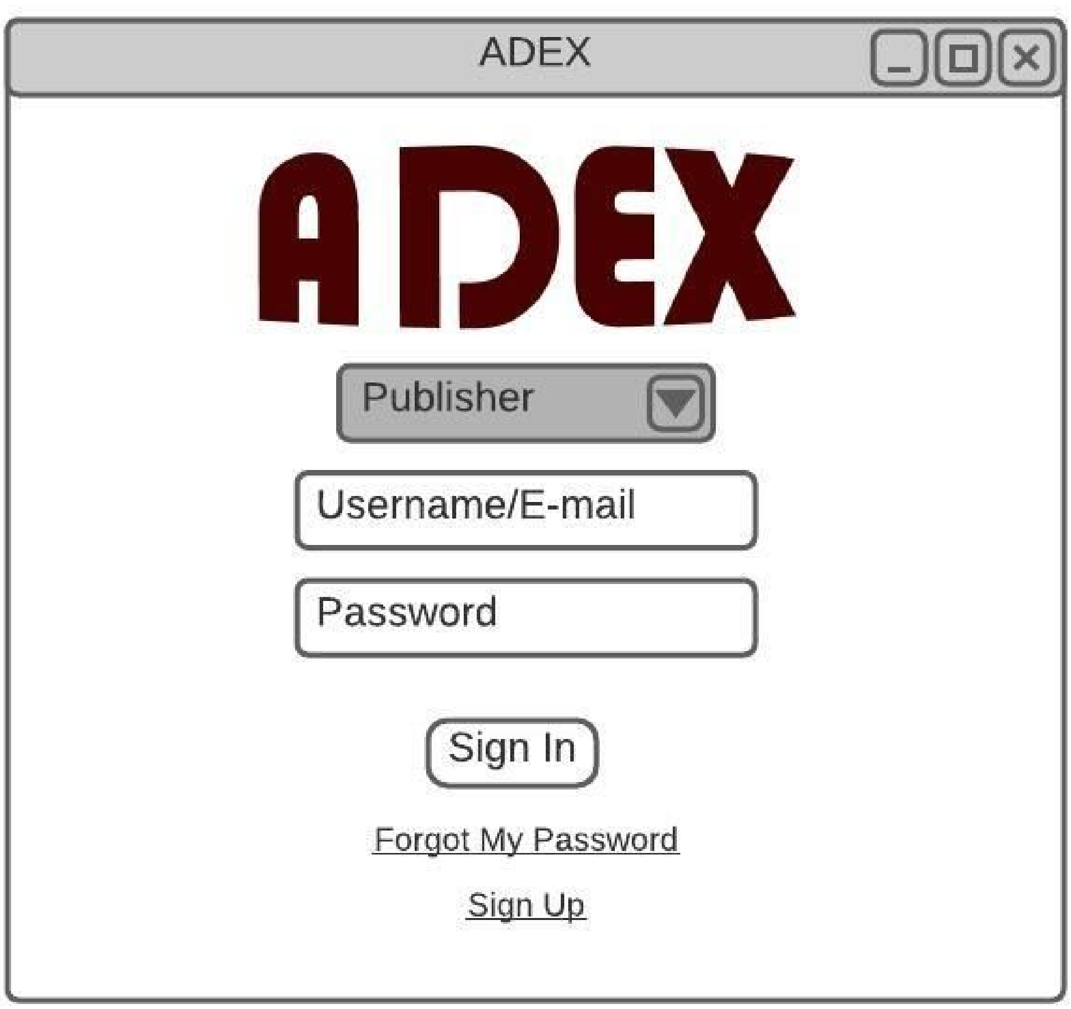
```
/*
Inputs: @curatorID
*/

SELECT U.username
FROM User U natural join Follow F natural join Curator C
WHERE C.curatorID = @curatorID
```

### Making a new Suggestion:

```
/*
Inputs: @curatorID, @gameID, @userID
*/

INSERT INTO Suggest(curatorID, gameID, usedID) VALUES (
        @curatorID,
        @gameID,
        @userID
);
```

## 5.3 Publisher Companies

### 5.2.1 Sign In Page



**Input:** @email_username, @password, @user_name, @email

**Process:** When the publisher companies open ADEX, they will be welcomed with our login page. If the publisher companies are not already signed up, they can sign up and go to the sign-up page after clicking the “Sign Up” link. Also, in the case of forgetting a password, publisher companies can change it through the “Forgot My Password” link. After logging in, the publisher companies will be directed to their published games page.

## SQL Statements:

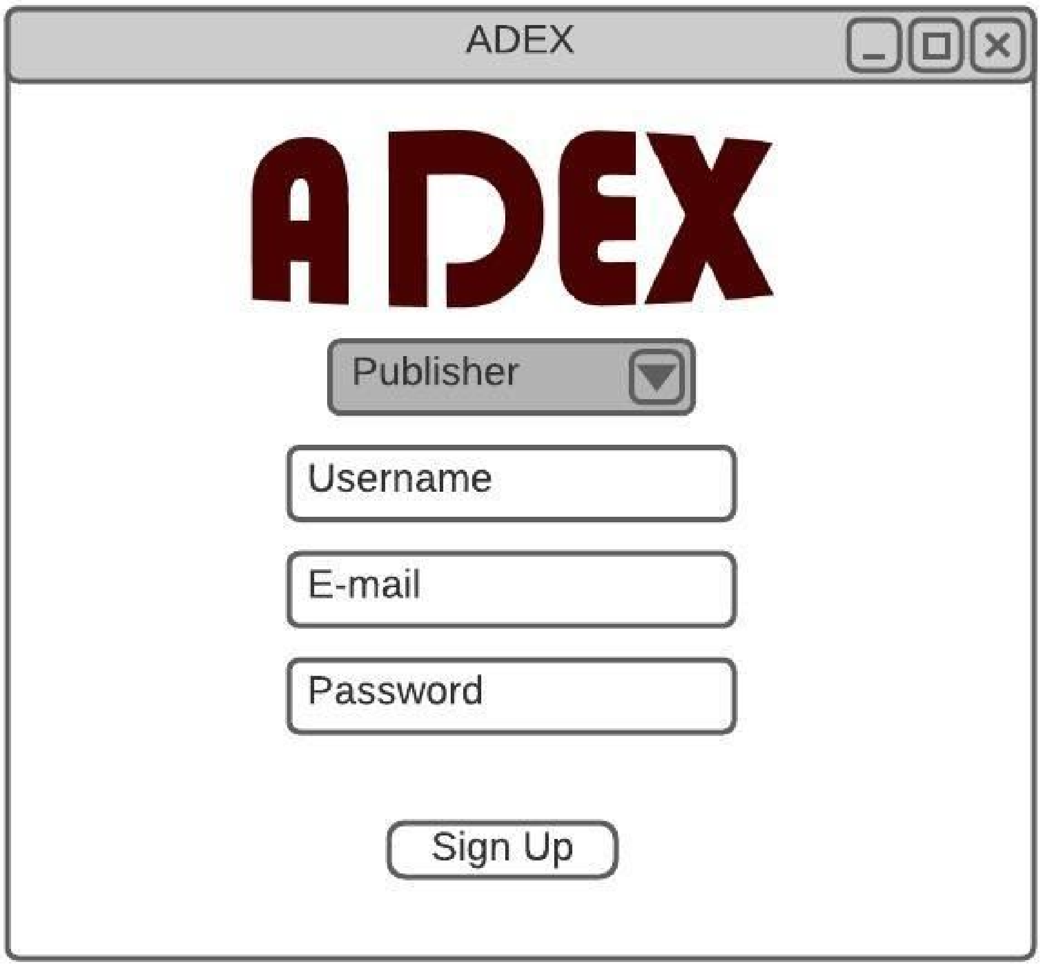
### Log - in:

```
/*
Inputs: @email_username, @password
*/

SELECT curatorID
FROM Publisher_Company
WHERE email = @email_username OR username = @email_username AND password =
@password
```

### Changing Password:

```
/*
Inputs: @publisherID, @password
*/

UPDATE Publisher_Company
SET password = @password
WHERE userID = @userID
```

### 5.2.2 Sign Up Page



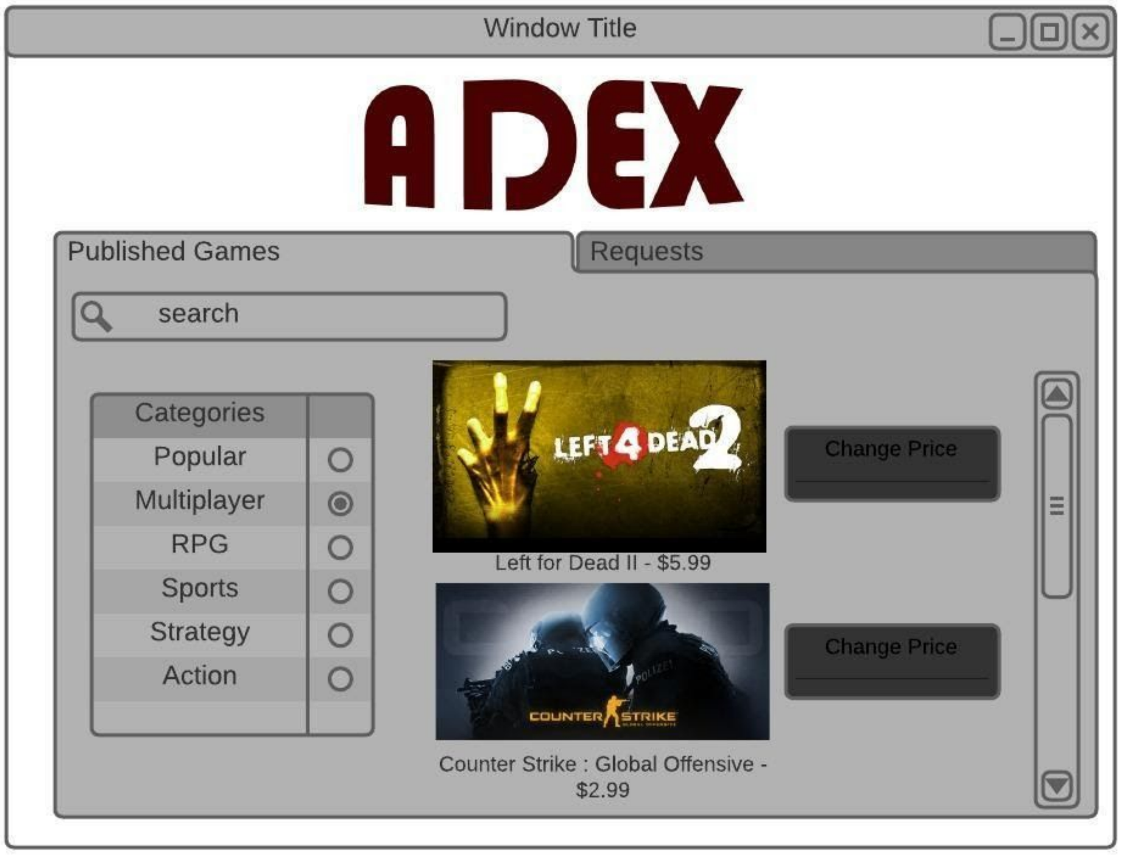
**Input:** @username, @email, @password

**Process:** When the publisher companies want to sign up, they will be directed to this page. After signing up, they will be directed to login page

**Sign up:**

```
/*
Inputs: @username, @email, @password
*/

INSERT INTO Publisher_Company(username, password, email) VALUES (
        @username,
        @password,
        @email
);
```

## 5.2.3 Published Games



**Input:** @publisherID

**Process:** In the published game section, publisher companies can see the games they already published. They can change the price of the games they want.

**SQL Statements:**

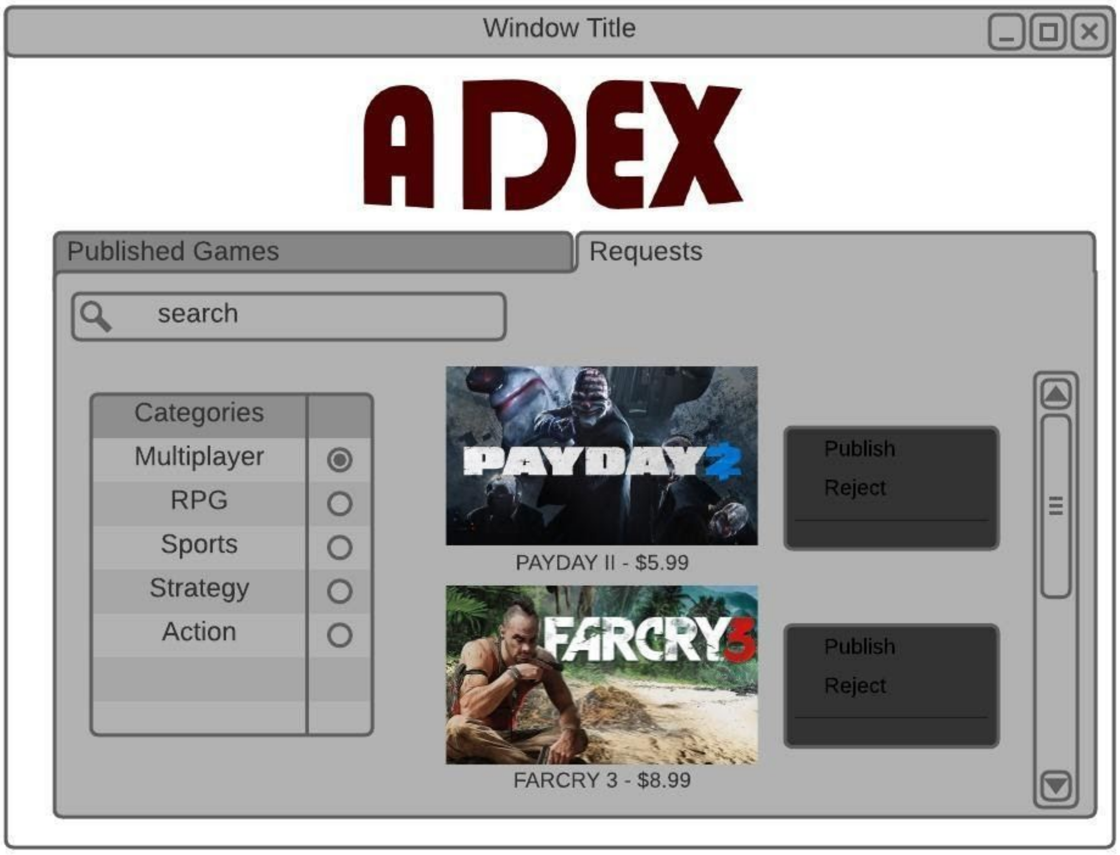
**Viewing Games:**

```
/*
Inputs: @publisherID, @genre,

*/

SELECT G.game_name, G.image,G.price
FROM Game G natural join Publisher_Company P natural join Publish PU
WHERE G.genre = @genre AND P.publisherID = @publisherID AND G.active = “Yes”
```

**Change the Price of the Game:**

```
/*
Inputs: @gameID, @n_price
*/

UPDATE Game
SET discount_amount = price - @n_price
WHERE gameID= @gameID

UPDATE Game
SET price = @n_price
WHERE gameID= @gameID
```

## 5.2.3 Requests



**Input:** @publisherID

**Process:** In the requests section, publisher companies can see the games requested by developer companies. They can either publish the game or reject it.

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @publisherID, @genre,

*/

SELECT G.game_name, G.image,G.price, D.username
FROM Game G natural join Publisher_Company P natural join Request natural join
Developer_Company
WHERE G.genre = @genre AND P.publisherID = @publisherID
;
```

**Publish a Game:**

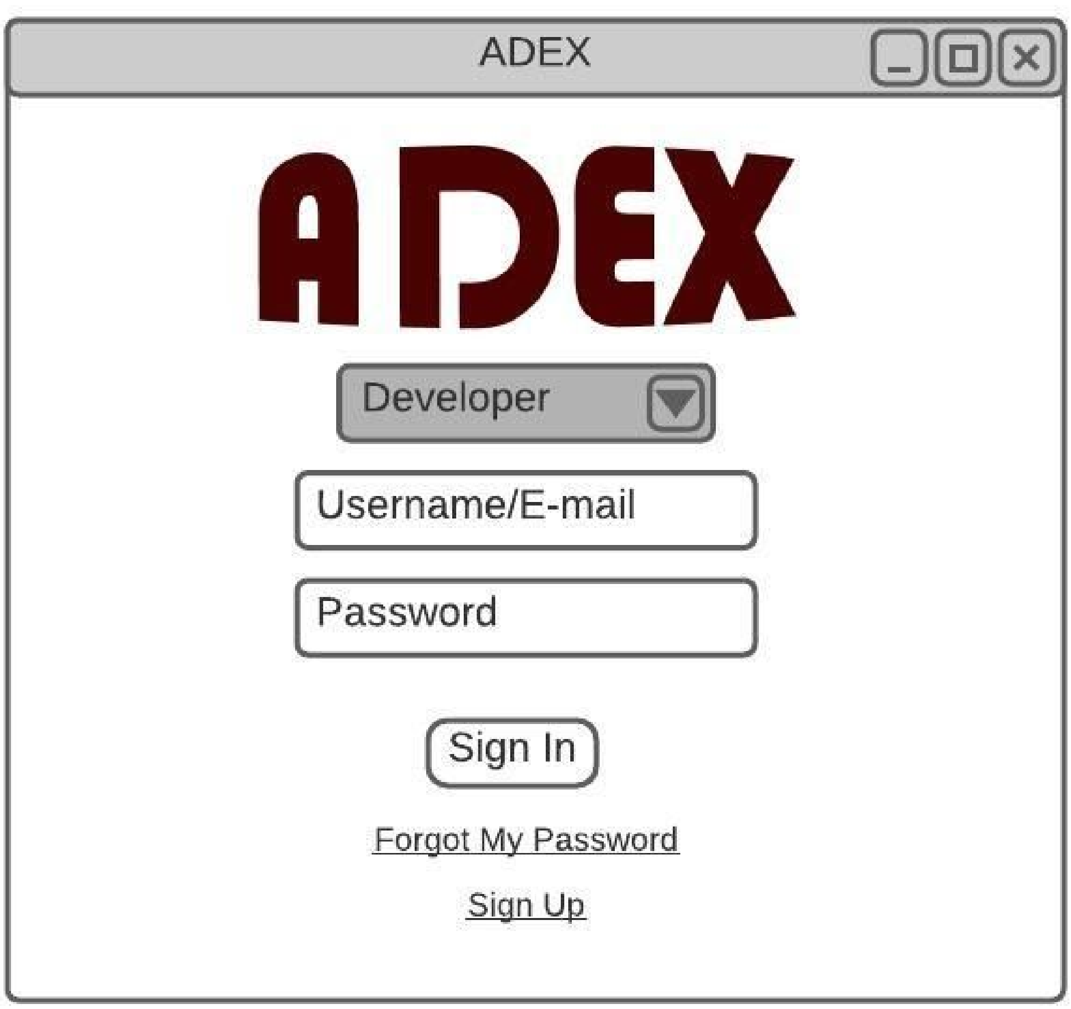
```
/*
Inputs: @publisherID, @gameID, @developerID

*/
INSERT INTO Publish(PublisherID, gameID, developerID) VALUES (
        @publisherID,
        @gameID,
        @developerID
);

UPDATE Game
WHERE gameID = @gameID
SET active=”Yes”
```

**Reject Request:**

```
/*
Inputs: @publisherID, @gameID, @developerID
*/

DELETE FROM Request(publisherID, developerID, gameID)
WHERE publisherID= @publisherID AND developerID= @developerID AND gameID=
@gameID
```

## 5.4 Developer Companies

### 5.2.1 Sign In Page



**Input:** @email_username, @password, @user_name, @email

**Process:** When the developer companies open ADEX, they will be welcomed with our login page. If the developer companies are not already signed up, they can sign up and go to the sign-up page after clicking the “Sign Up” link. Also, in the case of forgetting a password, developer companies can change it through the “Forgot My Password” link. After logging in, the developer companies will be directed to the published games section.

## SQL Statements:

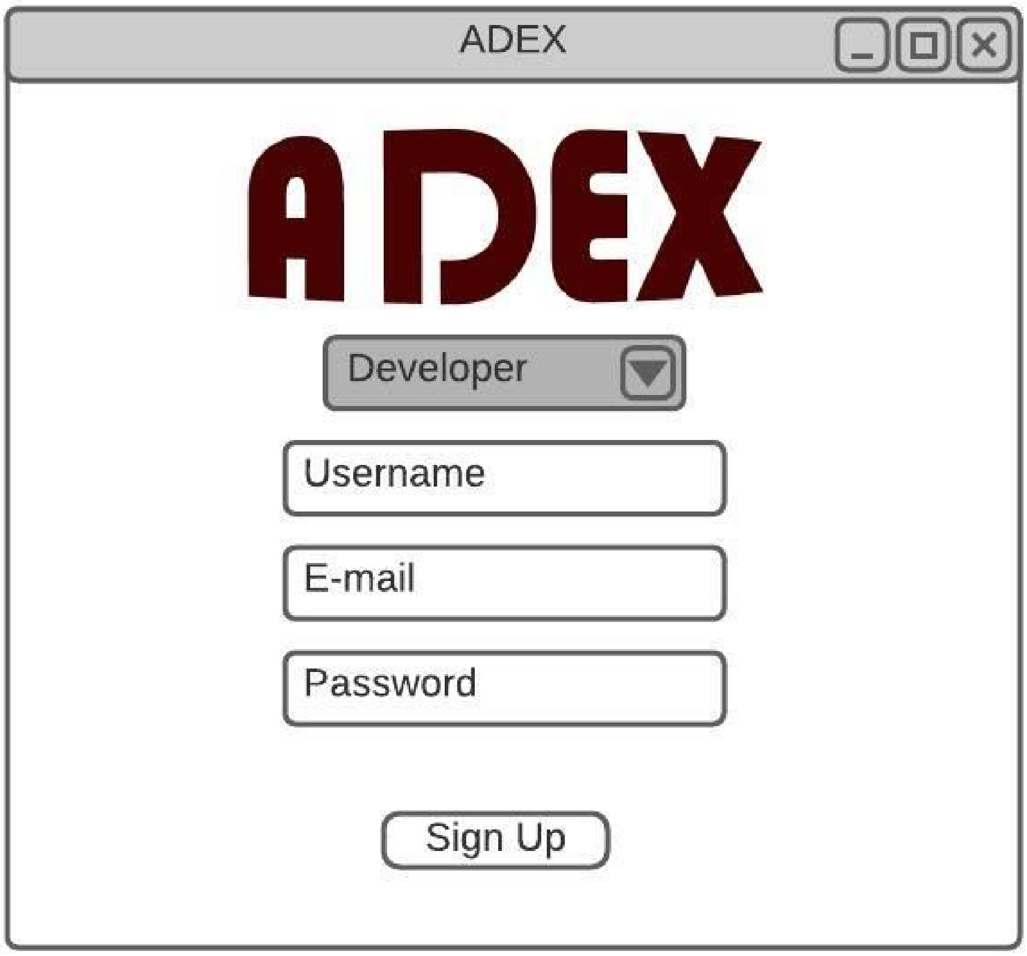
### Log - in:

```
/*
Inputs: @email_username, @password
*/

SELECT developerID
FROM Developer_Company
WHERE email = @email_username OR username = @email_username AND password =
@password
```

### Changing Password:

```
/*
Inputs: @developerID, @password
*/

UPDATE Developer_Company
SET password = @password
WHERE developerID = @developerID
```

## 5.2.2 Sign Up Page



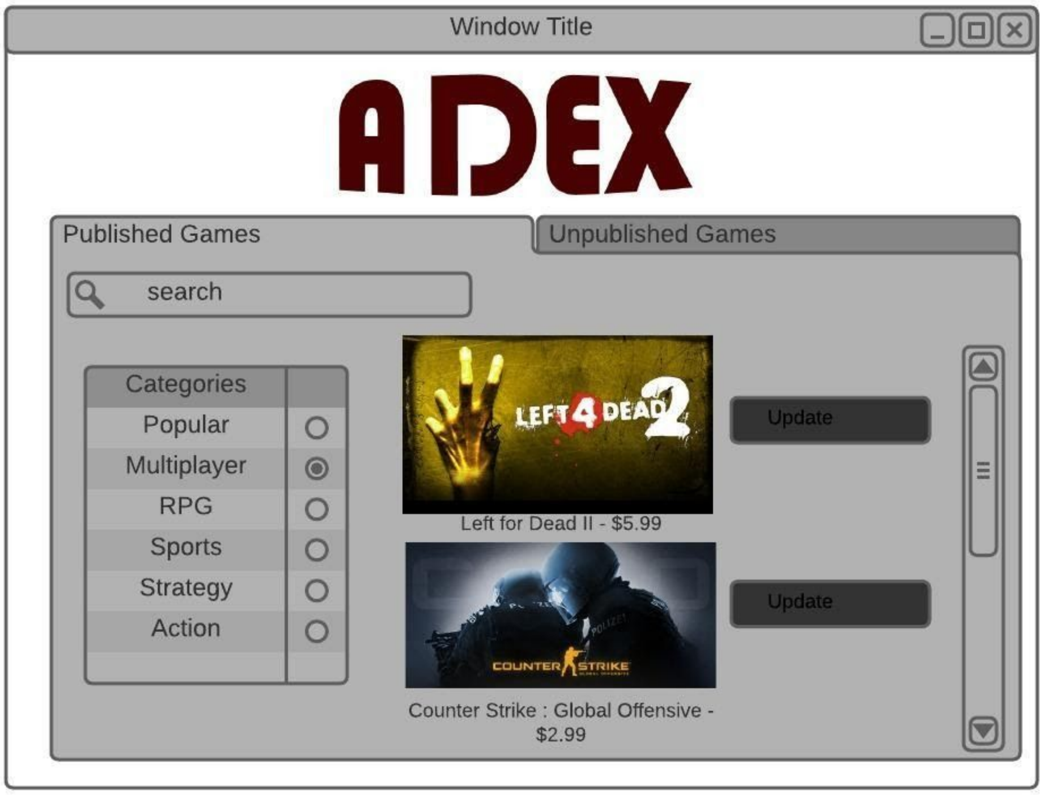
**Input:** @username, @email, @password

**Process:** When the developer companies want to sign up, they will be directed to this page. After signing up, they will be directed to login page

**Sign up:**

```
/*
Inputs: @username, @email, @password
*/

INSERT INTO Developer_Company(username, password, email) VALUES (
        @username,
        @password,
        @email
);
```

## 5.2.3 Published Games



**Input:** @developerID

**Process:** In the store section, developer companies can see the published games they developed. If they want to update the new game, they can click on the “Update” button.

**SQL Statements:**

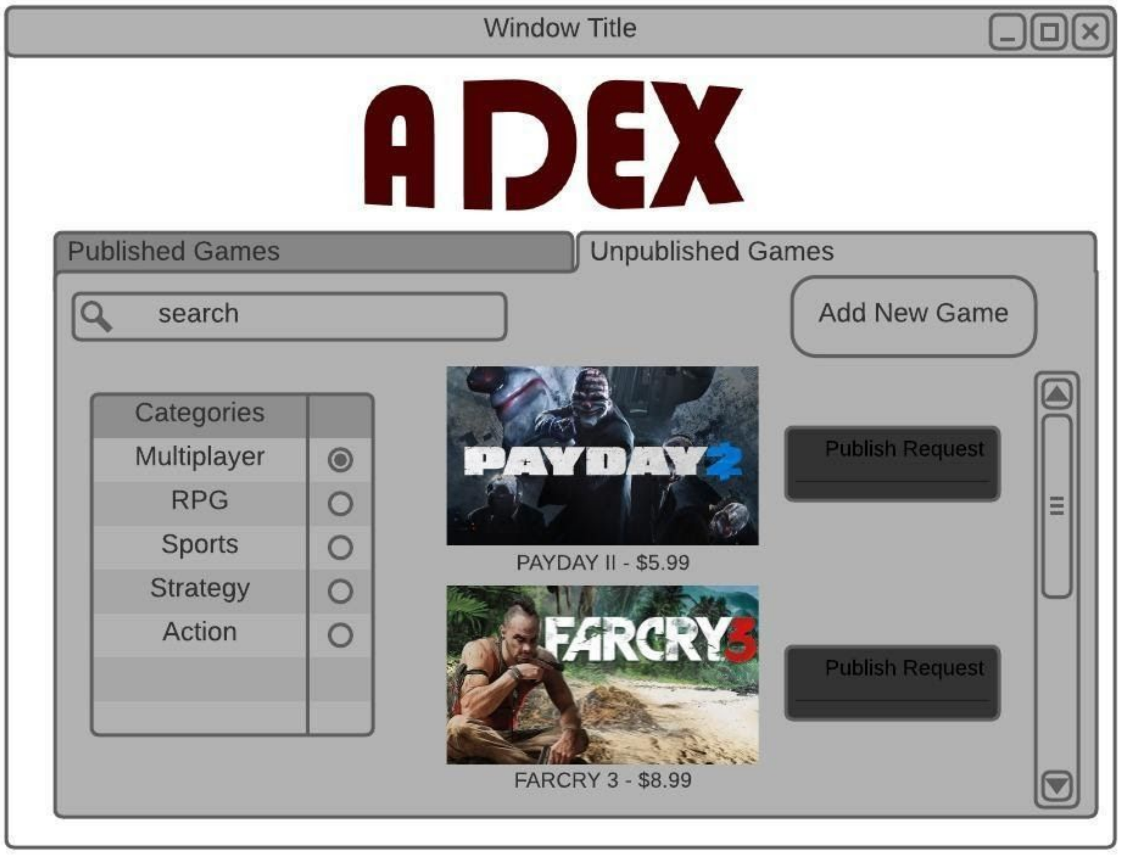
**Viewing Games:**

```
/*
Inputs: @developerID, @genre,

*/

SELECT G.game_name, G.image,G.price
FROM Game G natural join Publish P natural join Developer_Company D
WHERE G.genre = @genre AND D.developerID = @developerID AND G.active = “Yes”
```

**Updating Games:**

```
/*
Inputs: @developerID, @gameID, @date
*/

UPDATE Game
SET last_updated = date
WHERE gameID= @gameID

INSERT INTO Update(developerID, gameID) VALUES (
        @developerID,
        @gameID
);
```

### 5.2.3 Unpublished Games



**Input:** @developerID

**Process:** In the unpublished games section, developer companies can see the unpublished games they developed. If they want to add a new game, they can click the “Add New Game” button. If they want their games to publish they can click the “Publish Request” button and choose a publisher company to send a request.

**SQL Statements:**

**Viewing Games:**

```
/*
Inputs: @developerID, @genre,

*/

SELECT G.game_name, G.image,G.price
FROM Game G natural join Develop P natural join Developer_Company D
WHERE G.genre = @genre AND D.developerID = @developerID AND G.active = “No”
```

**Add a New Game:**

```
/*
Inputs: @developerID, @gameID, @game_name, @genre, @last_updated, @price,
@publish_date, @discount_amount, @image
*/

INSERT INTO Game(gameID, game_name, genre, last_updated, price, publish_date,
discount_amount, image, active) VALUES (
        @gameID,
        @game_name,
        @genre,
        @last_updated,
        @price,
        @publish_date,
        @discount_amount,
        @image,
        @active
);

INSERT INTO Develop(developerID, gameID) VALUES (
        @developerID,
        @gameID
);
```

**List all Publisher to Choose for a Request:**

```
SELECT username
FROM Publisher_Company
```

**Send Request:**

```
/*
Inputs: @developerID, @gameID, @publisherID
*/

INSERT INTO Request(publisherID, developerID, gameID) VALUES (
        @publisherID,
        @developerID,
        @gameID
);
```

# 6 Advanced Database Components

## 6.1 Reports

### 6.1.1 Total Number of Users

Total count of user data could be helpful in further implementation.

```
SELECT count(userID) AS totalUserCount
FROM User;
```

### 6.1.2 Total Number of Curators

Total count of curator data will have benefits in project development and production.

```
SELECT count(curatorID) AS totalCuratorCount
FROM Curator;
```

### 6.1.3 Total Number of Games

The quantity of games will be helpful in implementation of the project.

```
SELECT count(gameID) AS totalGameCount
FROM Game;
```

### 6.1.4 Total Number of Publisher Companies

The amount of publisher company data will be helpful in further implementation of the project.

```
SELECT count(publisherID) AS totalPublisherCount
FROM Publisher_Company;
```

### 6.1.5 Total Number of Developer Companies

Developer Company count will be good for understanding how much developers are there that develops games.

```
SELECT count(developerID) AS totalDeveloperCount
FROM Developer_Company;
```

### 6.1.6 Most Commented 5 Game

The users will use this SQL statement to see the most commented game in the store.

```
SELECT G.game_name,  count(C.gameID) as commentCount
FROM Game as G, Comment as C
WHERE G.gameID = C.gameID
ORDER BY commentCount DESC
LIMIT 5;
```

### 6.1.7 Most Reviewed 5 Game

The users will use this SQL statement to see the most reviewed game in the store.

```
SELECT G.game_name,  count(R.gameID) as reviewCount
FROM Game as G, Review as R
WHERE G.gameID = R.gameID
ORDER BY reviewCount DESC
LIMIT 5;
```

### 6.1.8 Total Number of Games Published by Each Publisher Company

The users will see the amount of games published by each publisher company in the publisher company’s page.

```
SELECT P1.publisherID, count(P2.gameID)
FROM Publisher_Company as P1, Publish as P2
WHERE P1.publisherID = P2.publisherID
GROUP BY P1.publisherID;
```

### 6.1.9 Total Number of Games Developed by Each Developer Company

The users will see the amount of games developed by each developer company in the developer company’s page.

```
SELECT D1.developerID, count(D2.gameID)
FROM Developer_Company as D1, Develop as D2
WHERE D1.develeperID = D2.developerID
GROUP BY D1.developerID;
```

### 6.1.10 Total Money Spent On Each Game

This SQL statement will be beneficial for sorting the games according to their price and the popularity.

```
SELECT G.game_name, count(L.gameID) * G.price
FROM Game as G, Library as L
WHERE L.gameID = G.gameID
GROUP BY G.game_name;
```

## 6.2 Views

### 6.2.1 Comments of a Game View

The view of all comments of a game.

```
CREATE VIEW commentsOfAGame AS
        SELECT G.game_name, U.username as username, C.text as comment
        FROM Game as G, Comment as C, User as U
        WHERE C.gameID = G.gameID and U.userID = C.userID
        ORDER BY C.date DESC;
```

### 6.2.2 Wishlists of Games of a User View

The view of all the games of a user’s wishlist.

```
CREATE VIEW wishlistGames AS
        SELECT G.game_name, G.image
        FROM Game as G, Wishlist as W, User as U
        WHERE W.gameID = G.gameID and U.userID = W.userID;
```

### 6.2.3 Games Categorised by Genre View

The view of games categorized by their genre.

```
CREATE VIEW categorisedGames AS
        SELECT game_name, image, genre
        FROM Game
        GROUP BY genre;
```

### 6.2.4 Games of a User in their Library View

The view of games of a user has in its library.

```
CREATE VIEW libraryGames AS
        SELECT G.game_name, G.image
        FROM Game as G, Library as L
        WHERE G.gameID = L.gameID;
```

### 6.2.5 Friends of a User View

The view of the friends a user has.

```
CREATE VIEW friends AS
        SELECT U2.username, U2.image
        FROM User as U1, User as U2, Friend as F
        WHERE U1.userID = F.userID1 and U2.userID = F.userID2;
```

### 6.2.6 Followed Curators of a User View

The view of the curator a user follows.

```
CREATE VIEW curators AS
        SELECT C.username
        FROM User as U, Curator as C, Follow as F
        WHERE U.userID = F.userID and C.curatorID = F.curatorID;
```

## 6.3 Triggers

1. When a user buys a game, the game will be added to its library, the balance of the user will decrease according to the game's price, and if the game is in the users wishlist relation, the game will be removed from the wishlist relation of the user.
2. When a user comments a game, the comments will be added to the comments relation.
3. When a user invites a friend to play a game, it will be added into the invitation relation.
4. When a user follows a curator, it will be added into the follow relation.
5. When a curator reviews a game, it will be added into the review relation.
6. When a user builds a mod for a game, the mod will be added into the build relation.
7. When a developer company develops a game, it will be added into the develop relation.
8. When a developer company updates a game, it will be added into the update relation and the last_update attribute of the game will be updated.

9. When a publisher company publishes a game, it will be added into the publish relation and publish_date of a game will be set.
10. When a user requests to be friends with a user, it will be added into friend_request relation.
11. When a user accepts to be friends with a user, it will be first removed from the friend_request relation and added into the friend relation while setting the date.
12. When a user accepts or declines a game invitation, it will be removed from the invitation relation.

## 6.4 Constraints

1. Users need to be logged in to perform any action.
2. Users need to have enough balance to buy a game.
3. Users can only send invitations to their friends.
4. Comments and reviews cannot be blank.
5. A developer company can only update its own game.
6. Users can only build mods for the games they own.
7. Users can only add games to their wishlist if they do not have the game.
8. Users can only comment on a game if they have the game.
9. Users can send friend requests to users if they have not requested a friend request before.
10. The names of games should be unique.

## 6.5 Stored Procedures

### 6.5.1 Buying a Game Procedure

This stored procedure will be used to add a game to a users library and update the users balance according to the price of the game.

```
DELIMITER $$
CREATE PROCEDURE buyGame (
        IN userID1 INT,
        IN gameID1 INT,
        IN gameMod VARCHAR(255),
        IN date1 DATE,
        IN balance1 INT,
        IN price1 INT
        )
        BEGIN
                IF (balance1 >= price1)
                        INSERT INTO Library(userID1, gameID1, mod1, date1);
                        UPDATE User
                        SET balance = balance - price1;
                        WHERE userID = userID1;
        END $$
DELIMETER ;
```

### 6.5.2 Commenting a Game Procedure

This stored procedure will be used to comment on a game.

```
DELIMITER $$
CREATE PROCEDURE commentGame (
        IN userID1 INT,
        IN gameID1 INT,
        IN text1 LONGTEXT,
        IN date1 DATE,
        IN rating INT
        )
        BEGIN
                INSERT INTO Comment(userID1, gameID1, date1, text1, rating);
                SELECT userID1, gameID1, date1, text1, rating
                FROM Library
                WHERE EXISTS
                        (SELECT gameID
                        FROM Library
                        WHERE userID1 = userID and gameID1 = gameID
                        )
        END $$
DELIMETER ;
```

### 6.5.3 Adding a Game to Wishlist Procedure

This stored procedure will be used to add a game to a users wishlist.

```
DELIMITER $$
CREATE PROCEDURE addGameWishlist (
        IN userID1 INT,
        IN gameID1 INT
        )
        BEGIN
                INSERT INTO Wishlist(userID1, gameID1);
        SELECT userID1, gameID1
        FROM Wishlist, Library
        WHERE NOT EXISTS
            (SELECT gameID
            FROM Wishlist
            WHERE userID1 = userID and gameID1 = gameID
            )
            and NOT EXISTS
            (SELECT gameID
            FROM Library
            WHERE userID1 = userID and gameID1 = gameID
            )
    END $$
DELIMETER ;
```

## 7 Implementation Plan

In the project, we will use React.js as the front-end framework. For the back-end, we will use PHP to connect our system to the database. As for the database, we will use MySQL server for database management.

## 8 Website

https://egehakankar.github.io/adex/