Bilkent University
Department of Computer Engineering

# CS353-Database Systems

# Term Project

*Video Game Digital Distribution Service (i.e. Steam)*

## Design Report

**Instructor**

Özgür Ulusoy

**Assigned TA**

Mustafa Can Çavdar

**Team Members**

Ali Taha Dinçer

İrem Ecem Yelkanat

Muhammed Naci Dalkıran

Sena Korkut

## Contents

## 1 Description

Online game selling platforms are commonly used by people to buy and download games on their computers. With the development of online game selling platforms, gamers have become able to buy newly released games without any overhead for waiting for that game to come to their nearest retailer store and download and play immediately as long as they have an internet connection. Some of them allow users to buy and download games on their computers and some of them allow their users to play any game they want if they pay monthly in a subscription.

Online game selling platforms are important for both gamers, developers and publishers in different manners. With online game selling platforms, gamers started to be able to buy and play their games anytime on any computer, individual developers started to have platforms to sell their games and publishers started to sell games faster without any concern.

The purpose of this project is to implement video game digital distribution service, i.e. Steam and this report considers the design process of the system.

## 2 Revised E/R Diagram

After the review and design process, the previous E/R diagram of the project has altered. Following changes are applied to the system:

- ➔ "Likes/dislikes" relation is removed from between User-Comment and User-Video Game entities. "Rates" relation is added with the value attribute only for Video Game entities.
- ➔ Ternary relationship "leaves" is turned into a "comments_on" relation between User and Video Game along with attributes date and text.
- ➔ "Install" relation is added between User and Video Game in addition to "buys" relation. Additionally, version_no attribute is added for that relation for changing the version in updates. "return" relation is deleted due to availability of deleting games from "buys" relation without creating an extra relation for returning.
- ➔ Ternary relation "builds" is turned into a binary relationship and becomes a strong relation between User and Mod.

- ➔ Mod was a weak entity, changed to a strong entity. “for_m” relation is added between Mod and Video Game.
- ➔ “Suggest” relation is removed as Curator also has a review option.
- ➔ Attributes Text and Date are added to review relation.
- ➔ Attributes holding numbers such as num_of_followers, num_of_like are removed to be counted from relation tables.
- ➔ game_publisher and game_developer attributes are deleted from the Video Game entity as publisher and developer of a game can be found from “develops” and “publish” relations. Additionally, rank() and g_size attributes are deleted from Video Game. Instead, rank will be calculated from relation tables. requirements attribute is added to Video Game so that minimum system requirements and size of the game can be seen.
- ➔ u_ID attribute is removed and a_ID attribute is added to Account Entity as primary key. user_image is also removed for simplicity.
- ➔ Overlapping relation of User and Company to Account is changed to disjoint relation.
- ➔ Wallet and Credit Card entities are added along with “include” relation and weak “has” relation.
- ➔ New attributes are added to “updates” relation such as version_no and description of update.
- ➔ “develops” relation is added between Video Game and Developer Company.
- ➔ “permits” relation is removed. Instead, a Request entity is added along with the relations “asks”, “takes” and “about”.
- ➔ As an extra feature, Subscription Package is added. Package contains Video Games and a User can subscribe to this package.

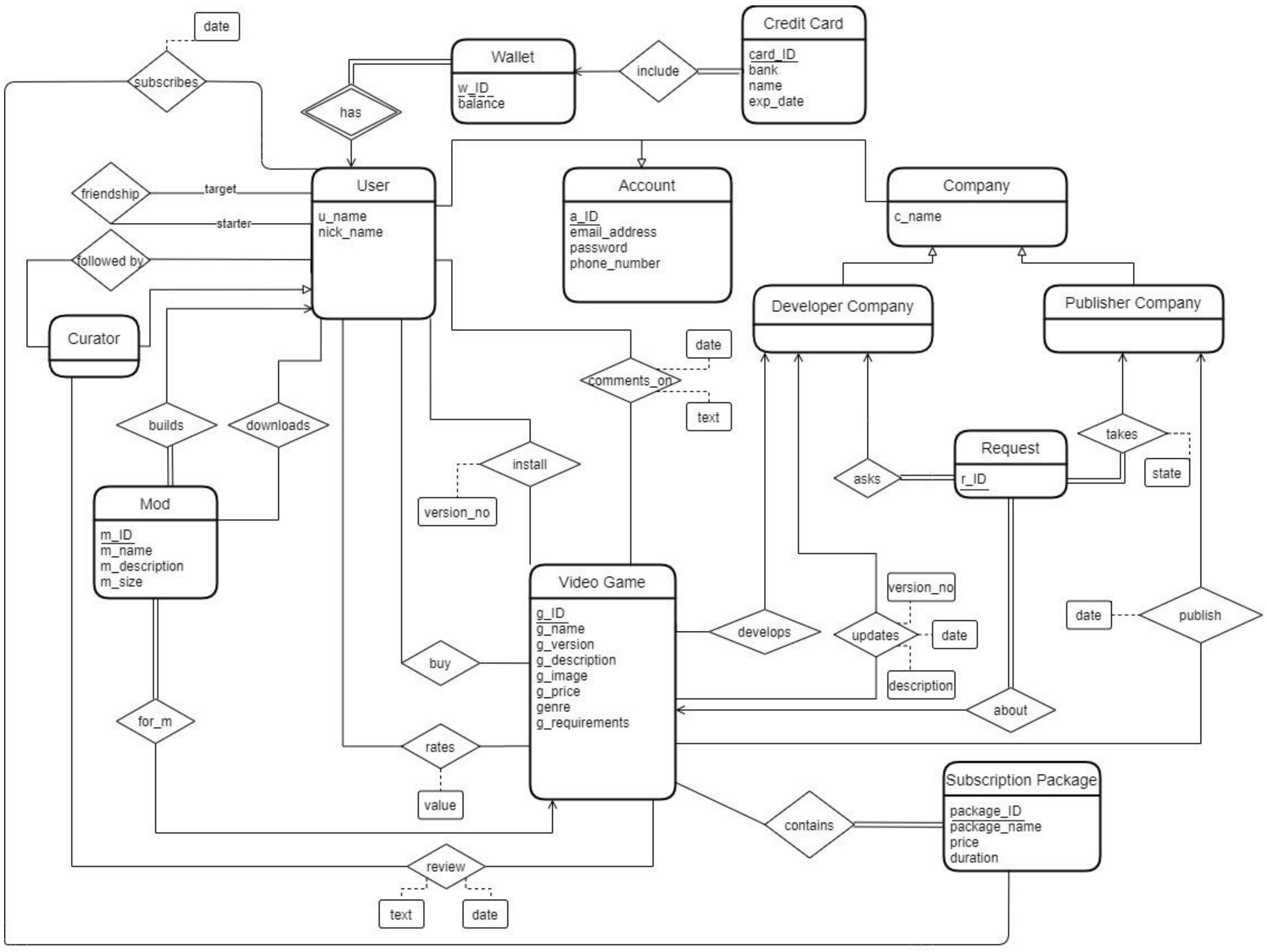


*Figure 1: Revised E/R Diagram of Video Game Digital Distribution Platform*

# 3 Relation Schemas

## 3.1 Account

### Relational Model

Account(a_ID, email_adress, password, phone_number)

### Functional Dependencies

a_ID -> email_address, password, phone_number

email_address -> a_ID, phone_number

phone_number -> a_ID, email_address

### Candidate Keys

{(a_ID), (email_adress)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Account(

a_ID INT AUTO_INCREMENT,

email_address VARCHAR(64) NOT NULL UNIQUE,

phone_number VARCHAR(15) UNIQUE,

password VARCHAR(32) NOT NULL,

PRIMARY KEY( a_ID )

) ENGINE = INNODB;
```

## 3.2 User

**Relational Model**

User(<u>a_ID</u>, u_name, nick_name)

**Functional Dependencies**

a_ID -> u_name, nick_name

nick_name ->a_ID

**Candidate Keys**

{(a_ID), (nick_name)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE User(
a_ID INT,
u_name CHAR(30)  NOT NULL,
nick_name CHAR(15) NOT NULL UNIQUE,
PRIMARY KEY ( a_ID ),
FOREIGN KEY ( a_ID ) REFERENCES Account ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.3 Company

**Relational Model**

Company(<u>a_ID</u>,c_name)

**Functional Dependencies**

a_ID -> c_name

c_name -> a_ID

**Candidate Keys**

{(a_ID), (c_name)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Company(
a_ID INT,
c_name CHAR(30)  NOT NULL UNIQUE,
PRIMARY KEY ( a_ID ),
FOREIGN KEY ( a_ID ) REFERENCES Account ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.4 Curator

**Relational Model**

Curator(<u>a_ID</u>)

**Functional Dependencies**

-

**Candidate Keys**

{(a_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Curator(
a_ID INT,
PRIMARY KEY ( a_ID ),
FOREIGN KEY ( a_ID ) REFERENCES User ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.5 Developer Company

**Relational Model**

Developer_Company(a_ID)

**Functional Dependencies**

-

**Candidate Keys**

{(a_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Developer_Company(
a_ID INT PRIMARY KEY,
PRIMARY KEY ( a_ID ),
FOREIGN KEY ( a_ID ) REFERENCES Company ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.6 Publisher Company

**Relational Model**

Publisher_Company(<u>a_ID</u>)

**Functional Dependencies**

-

**Candidate Keys**

{(a_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Publisher_Company(
a_ID INT,
PRIMARY KEY ( a_ID ),
FOREIGN KEY ( a_ID ) REFERENCES Company ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.7 Wallet

**Relational Model**

Wallet(a_ID,w_ID, balance)

**Functional Dependencies**

a_ID, w_ID -> balance

**Candidate Keys**

{(a_ID, w_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Wallet(
a_ID INT,
w_ID INT AUTO_INCREMENT,
balance INT DEFAULT 0,
PRIMARY KEY ( a_ID, w_ID ),
FOREIGN KEY ( a_ID ) REFERENCES User ( a_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.8 Credit Card

**Relational Model**

Credit Card(card_ID, bank, name, exp_date)

**Functional Dependencies**

card_ID -> bank, name, exp_date

**Candidate Keys**

{(card_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Credit_Card(

card_ID CHAR(16),

bank VARCHAR(20),

name VARCHAR(50),

exp_date DATE NOT NULL,

PRIMARY KEY ( card_ID ),

) ENGINE = INNODB;
```

## 3.9 Video Game

**Relational Model**

Video Game(g_id, g_name, g_version, g_description, g_image, g_price, genre, g_requirements)

**Functional Dependencies**

g_ID -> g_name, g_version, g_description, g_image, g_price, genre, g_requirements

g_name -> g_ID

**Candidate Keys**

{(g_ID), (g_name)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Video_Game(
g_ID INT AUTO_INCREMENT,
g_name VARCHAR(20) NOT NULL UNIQUE,
g_version VARCHAR(5) DEFAULT 1.0.0,
g_description VARCHAR(280) NOT NULL,
g_image VARBINARY(512),
g_price INT DEFAULT 0,
genre VARCHAR(15),
g_requirements  VARCHAR(280) NOT NULL,
PRIMARY KEY ( g_ID )
) ENGINE = INNODB;
```

## 3.10 Mod

**Relational Model**

Mod(m_ID, m_name, m_description, m_size)

**Functional Dependencies**

m_ID -> m_name, m_description, m_size

m_name -> m_ID

**Candidate Keys**

{(m_ID), (m_name)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Mod(
m_ID INT AUTO_INCREMENT,
m_name VARCHAR(20) NOT NULL UNIQUE,
m_description VARCHAR(280) NOT NULL,
m_size INT NOT NULL DEFAULT 0,
PRIMARY KEY ( m_ID )
) ENGINE = INNODB;
```

## 3.11 Request

**Relational Model**

Request(r_ID)

**Functional Dependencies**

-

**Candidate Keys**

{(r_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Request(
r_ID INT PRIMARY KEY AUTO_INCREMENT
) ENGINE = INNODB;
```

## 3.12 Subscription Package

**Relational Model**

Subscription_Package(package_ID, package_name, price, duration)

**Functional Dependencies**

package_ID -> package_name, price, duration

**Candidate Keys**

{(package_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE Subscription_Package(
package_ID INT PRIMARY KEY AUTO_INCREMENT,
package_name VARCHAR(30) NOT NULL,
price INT DEFAULT 0,
duration TIME
) ENGINE = INNODB;
```

## 3.13 Subscribes

**Relational Model**

subscribes(a_ID, package_ID, date)

**Functional Dependencies**

a_ID, package_ID -> date

**Candidate Keys**

{(a_ID, package_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE subscribes(
a_ID INT,
package_ID INT,
date DATE,
PRIMARY KEY (a_ID, package_ID),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (package_ID) REFERENCES Subscription_Package(package_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.14 Followed By

**Relational Model**

followed by(c_ID, a_ID)

**Functional Dependencies**

-

**Candidate Keys**

{(c_ID, a_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE followed_by(
c_ID INT,
a_ID INT,
PRIMARY KEY (c_ID, a_ID),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (c_ID) REFERENCES Curator(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.15 Friendship

**Relational Model**

friendship(starter, target)

**Functional Dependencies**

-

**Candidate Keys**

{(starter, target)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE friendship(
starter INT,
target INT,
PRIMARY KEY (starter, target),
FOREIGN KEY (starter) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (target) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.16 Review

**Relational Model**

review(c_ID, g_ID, text, date)

**Functional Dependencies**

c_ID, g_ID -> text, date

**Candidate Keys**

{(c_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE review(
c_ID INT,
g_ID INT,
text VARCHAR(140) NOT NULL,
date DATE,
PRIMARY KEY (c_ID, g_ID),
FOREIGN KEY (c_ID) REFERENCES Curator(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.17 Builds

**Relational Model**

builds(m_ID, a_ID)

**Functional Dependencies**

m_ID -> a_ID

**Candidate Keys**

{(m_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE builds(
m_ID INT,
a_ID INT,
PRIMARY KEY (m_ID),
FOREIGN KEY (m_ID) REFERENCES Mod(m_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.18 Downloads

**Relational Model**

downloads(m_ID, a_ID)

**Functional Dependencies**

-

**Candidate Keys**

{(m_id, a_id)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE downloads(
m_ID INT,
a_ID INT,
PRIMARY KEY (m_ID, a_ID),
FOREIGN KEY (m_ID) REFERENCES Mod(m_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.19 For_m

**Relational Model**

for_m(m_ID, g_ID)

**Functional Dependencies**

m_ID -> g_ID

**Candidate Keys**

{(m_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE for_m(
m_ID INT,
g_ID INT,
PRIMARY KEY (m_ID),
FOREIGN KEY (m_ID) REFERENCES Mod(m_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.20 Rates

**Relational Model**

rates(a_ID, g_ID,value)

**Functional Dependencies**

a_ID, g_ID -> value

**Candidate Keys**

{(a_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE rates(
a_ID INT,
g_ID INT,
value INT,
PRIMARY KEY (a_ID, g_ID),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.21 Buys

**Relational Model**

buys(a_ID, g_ID, date)

**Functional Dependencies**

a_ID, g_ID -> date

**Candidate Keys**

{(a_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE buys(
a_ID INT,
g_ID INT,
date DATE,
PRIMARY KEY (a_ID, g_ID),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.22 Install

**Relational Model**

install(a_ID, g_ID, version_no)

**Functional Dependencies**

a_ID, g_ID -> version_no

**Candidate Keys**

{(a_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE install(
a_ID INT,
g_ID INT,
version_no VARCHAR(15),
PRIMARY KEY (a_ID, g_ID),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.23 comments_on

**Relational Model**

comments_on(a_ID, g_ID, date, text)

**Functional Dependencies**

a_ID, g_ID -> date, text

**Candidate Keys**

{(a_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE comments_on(
a_ID INT,
g_ID INT,
date DATE NOT NULL,
text VARCHAR(140) NOT NULL,
PRIMARY KEY ( a_ID, g_ID ),
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY ( g_ID ) REFERENCES Video_Game ( g_ID )
        ON DELETE CASCADE
        ON UPDATE RESTRICT
) ENGINE = INNODB;
```

## 3.24 Include

**Relational Model**

include(card_ID,w_ID, a_ID)

**Functional Dependencies**

card_ID -> w_ID,  a_ID

**Candidate Keys**

{(card_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE include(
card_ID INT,
w_ID INT,
a_ID INT,
PRIMARY KEY (card_ID),
FOREIGN KEY (card_ID) REFERENCES Credit_Card(card_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (w_ID) REFERENCES Wallet(w_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES User(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.25 Develops

**Relational Model**

develops(a_ID, g_ID)

**Functional Dependencies**

g_ID -> a_ID

**Candidate Keys**

{(g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE develops(
g_ID INT,
a_ID INT ,
PRIMARY KEY  (g_ID),
FOREIGN KEY (a_ID) REFERENCES Developer_Company(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.26 Updates

**Relational Model**

updates(a_ID, g_ID, date, version_no, description)

**Functional Dependencies**

g_ID -> a_ID, date, version_no, description

**Candidate Keys**

{(g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE updates(
g_ID INT,
a_ID INT,
date DATE,
version_no VARCHAR(15),
description VARCHAR(140),
PRIMARY KEY  (g_ID),
FOREIGN KEY (a_ID) REFERENCES Developer_Company(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.27 Contains

**Relational Model**

contains(package_ID, g_ID)

**Functional Dependencies**

-

**Candidate Keys**

{(package_ID, g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE contains(
package_ID INT,
g_ID INT,
PRIMARY KEY  (package_ID, g_ID),
FOREIGN KEY (package_ID) REFERENCES Subscription_Package(package_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.28 Asks

**Relational Model**

asks(r_ID, a_ID)

**Functional Dependencies**

r_ID -> a_ID

**Candidate Keys**

{(r_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE asks(
r_ID INT,
a_ID INT,
PRIMARY KEY  (r_ID),
FOREIGN KEY (r_ID) REFERENCES Request(r_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES Developer_Company(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.29 About

**Relational Model**

about(r_ID, g_ID)

**Functional Dependencies**

r_ID -> g_ID

**Candidate Keys**

{(r_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE about(
r_ID INT,
g_ID INT,
PRIMARY KEY  (r_ID),
FOREIGN KEY (r_ID) REFERENCES Request(r_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.30 Publish

**Relational Model**

publish(g_ID, a_ID, date)

**Functional Dependencies**

g_ID -> a_ID, date

**Candidate Keys**

{(g_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE publish(
g_ID INT,
a_ID INT,
date DATE,
PRIMARY KEY  (g_ID),
FOREIGN KEY (g_ID) REFERENCES Video_Game(g_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES Publisher_Company(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 3.31 Takes

**Relational Model**

takes(r_ID, a_ID, state)

**Functional Dependencies**

r_ID -> a_ID, state

**Candidate Keys**

{(r_ID)}

**Normal Form**

BCNF

**Table Definition**

```
CREATE TABLE takes(
r_ID INT,
a_ID INT,
state VARCHAR(8),
PRIMARY KEY  (r_ID),
FOREIGN KEY (r_ID) REFERENCES Request(r_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
FOREIGN KEY (a_ID) REFERENCES Publisher_Company(a_ID)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
) ENGINE = INNODB;
```

## 4 Normalization of Tables

As stated in the previous section, all of the tables created are in the Boyce Codd Normal Form (BCNF). It can be checked by the left-hand side of the dependencies, as they are either trivial or a super-key in the relation. Hence, decomposition or normalization is not needed.

## 5 User Interface Design and Corresponding SQL Queries

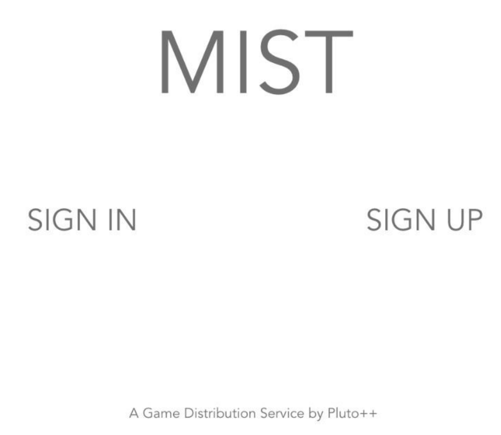


*Figure 2: Opening page of the application*

When the application is opened, the opening page in *Figure 2* will be displayed.

## 5.1 Sign Up

**User Interface:**

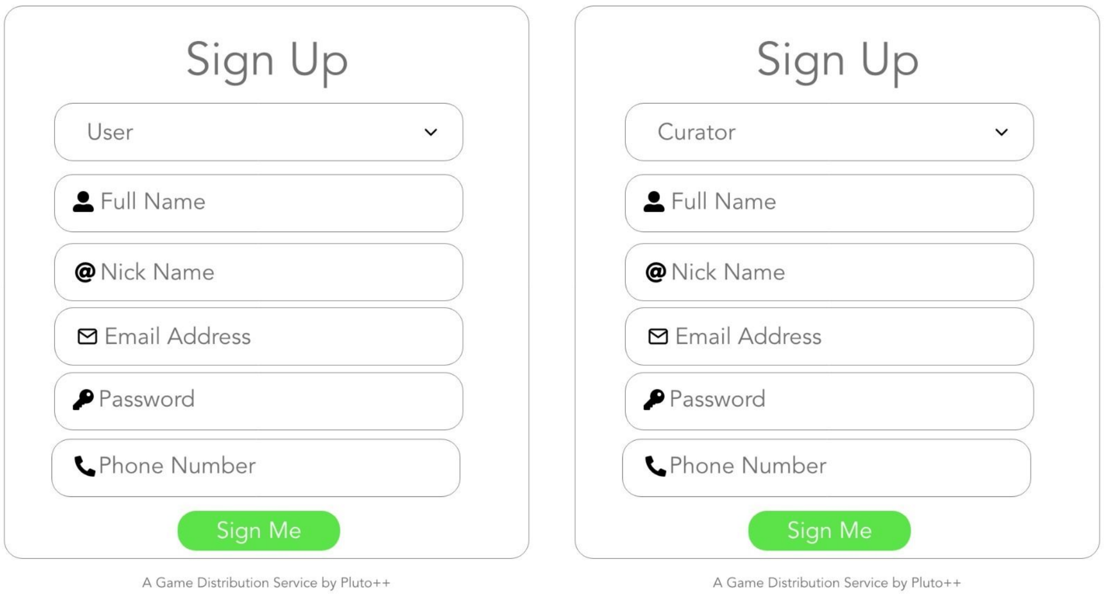


*Figure 3: Sign Up Screens for Users (User and Curator)*

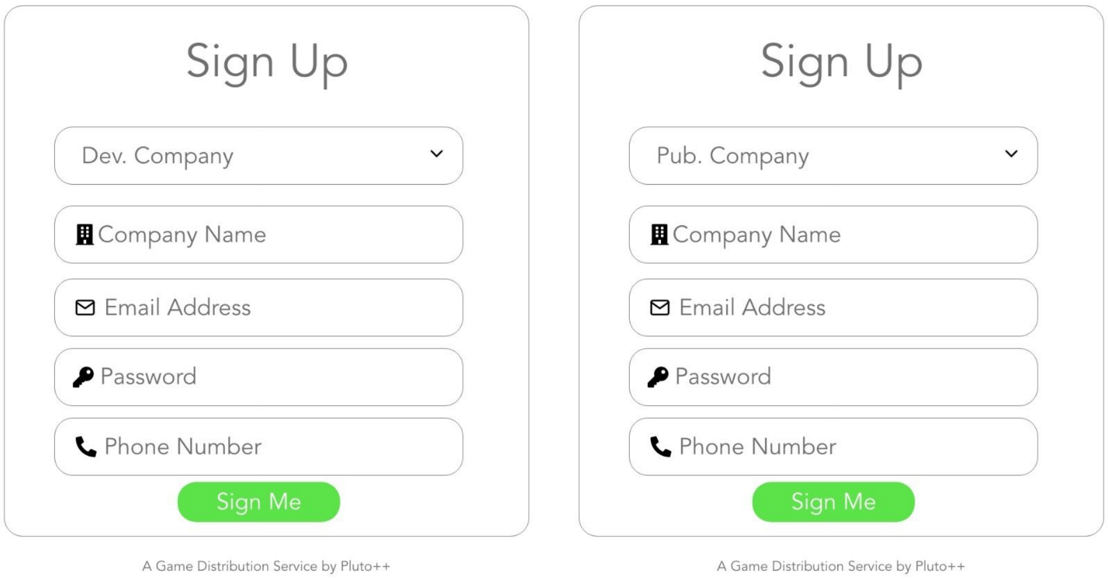


*Figure 4: Sign Up Screens for Companies(Developer and Publisher Company)*

In sign up screen, user will first choose the type of the account, i.e. User, Curator, Developer Company and Publisher Company. If the account is a type of User, full name and nickname along with e-mail address, password and phone number will be asked. If the account is a type of Company, company name along with e-mail address, password and phone number will be asked.

**Corresponding SQL Statements:**

Values taken from the user are e-mail address as @email_address, password as @password, phone number as @phone_number, user name as @u_name, nick name as @nick_name. Values taken from the company are e-mail address as @email_address, password as @password, phone number as @phone_number and company name as @c_name.

- The existence of @email_address and @phone_num in the database will be checked.

```
SELECT *

FROM Account

WHERE phone_num = @phone_num;

SELECT *

FROM Account

WHERE email_address = @email_address;
```

- For user, the existence of @nick_name and for the company, the existence of @c_name in the database will be checked.

```
SELECT *

FROM User

WHERE nick_name = @nick_name;



SELECT *

FROM Company

WHERE c_name = @c_name;
```

- ➔ The account will be added to the Account database if the values email and phone number are unique for both User and Account, if the value nickname is unique for User and  if the value company name is unique for Company.

```
INSERT INTO Account(email_address, password, phone_number)

VALUES (@email_address, @password, @phone_number);
```

- ➔ According to whether the account is a User or Company account, data will also be stored in either User or Company tables.
- ➔ First, with the given email, a_ID field from the Account table will be selected.

```
SELECT acc_ID

FROM Account

WHERE email_address = @email_address;
```

- ➔ With the selected a_ID as acc_ID, the values will be added to the tables according to the selected user types from figures Figure 3 and Figure 4.

```
INSERT INTO User

VALUES (acc_ID, @u_name, @nick_name);



INSERT INTO Curator

VALUES (acc_ID);



INSERT INTO Company

VALUES (acc_ID, @c_name);



INSERT INTO Developer_Company

VALUES (acc_ID);
```

```
INSERT INTO Publisher_Company

VALUES (acc_ID);
```

➔ If the type is a user, a wallet for that user will be created.

```
INSERT INTO Wallet(a_ID, balance)

VALUES (acc_ID, 0);
```

## 5.2 Sign In

**User Interface:**

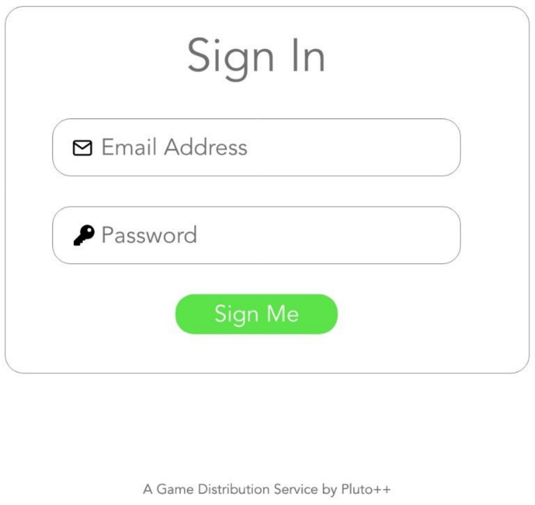


*Figure 5: Sign in Screen for all user types*

**Corresponding SQL Statements:**

Values taken from the user are e-mail address as @email_address, password as @password.

➔ Given e-mail address and password is checked from the Account table whether the account exists.

```
SELECT *

FROM Account

WHERE email_address = @email_address AND password = @password;
```

If an Account with entered attributes does not exist, the system will display a warning message.

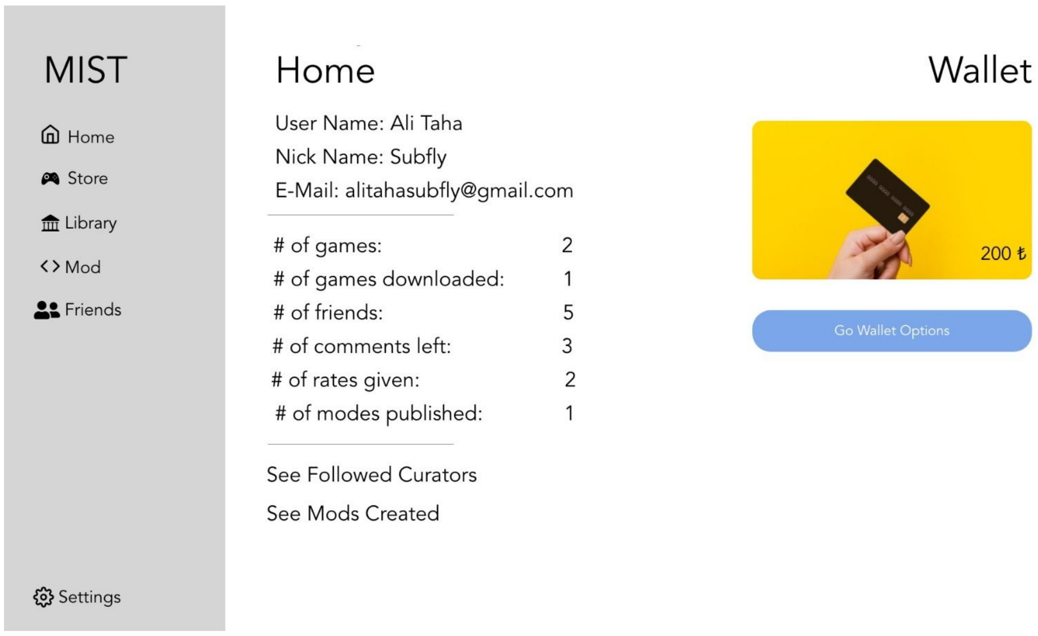
➔ After finding the corresponding account, account type should be checked.

```
SELECT acc_ID

FROM Account

WHERE email_address = @email_address;


SELECT *

FROM Curator

WHERE a_ID = acc_ID;


SELECT *

FROM Developer_Company

WHERE a_ID = acc_ID;


SELECT *

FROM Publisher_Company

WHERE a_ID = acc_ID;
```

## 5.3 Home Page

### 5.3.1 User

**User Interface:**



Home

User Name: Ali Taha
Nick Name: Subfly
E-Mail: alitahasubfly@gmail.com

| | |
|---|---|
| # of games: | 2 |
| # of games downloaded: | 1 |
| # of friends: | 5 |
| # of comments left: | 3 |
| # of rates given: | 2 |
| # of modes published: | 1 |

See Followed Curators
See Mods Created

Wallet

200 ₺

*Figure 6: Home page for User*

In the home page, shown above *Figure 6*, several different information about the user and the activity is shown along with the wallet and its balance. From the left side, the user can change between pages for different procedures.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Home page for the user contains user information namely the user's name, nick name, email.

```
SELECT u_name, nick_name, email_address

FROM User  NATURAL JOIN Account

WHERE a_ID = @a_ID;
```

- ➔ Home page for the user contains the number of games the user has, games downloaded, friends, comments left, rates given and modes published.

```
SELECT COUNT(*) AS num_of_games

FROM buy b

WHERE b.a_ID = @a_ID;


SELECT COUNT(*) AS num_of_games_downloaded

FROM install i, User u,

WHERE i.a_ID = @a_ID AND i.a_ID = u.a_ID;


SELECT COUNT(*) AS num_of_friends,

FROM friendship f

WHERE f.starter =@a_ID OR f.target =@a_ID;


SELECT COUNT(*) AS num_of_comments

FROM comments_on c

WHERE c.a_ID =@a_ID;


SELECT COUNT(*) AS num_of_rates_given,

FROM rates r

WHERE r.a_ID =@a_ID;


SELECT COUNT(*) AS num_of_mode_published

FROM builds b
```

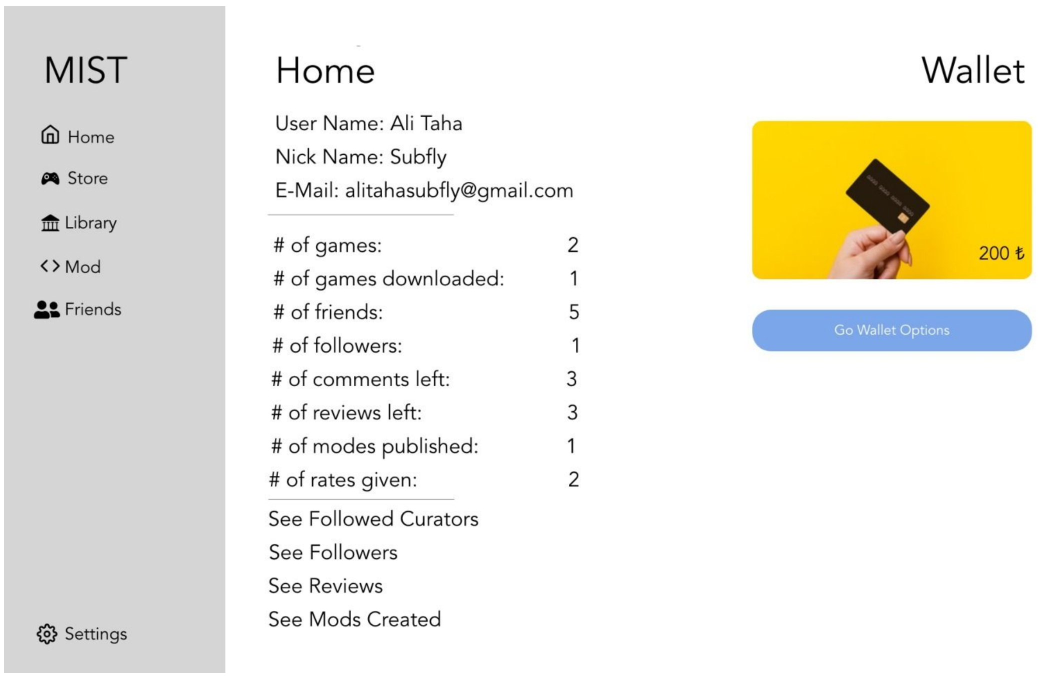
```
WHERE b.a_ID =@a_ID;
```

➔ Home page for the user contains the user's wallet and its balance information .

```
SELECT w.balance

FROM Wallet w

WHERE w.a_ID = @a_ID;
```

### 5.3.2 Curator

**User Interface:**



User Name: Ali Taha
Nick Name: Subfly
E-Mail: alitahasubfly@gmail.com

| | |
|---|---|
| # of games: | 2 |
| # of games downloaded: | 1 |
| # of friends: | 5 |
| # of followers: | 1 |
| # of comments left: | 3 |
| # of reviews left: | 3 |
| # of modes published: | 1 |
| # of rates given: | 2 |

*Figure 7: Home Page for Curator*

For a curator, the user interface for the home page is changed slightly as in *Figure 7*.

**Corresponding SQL Statements:**

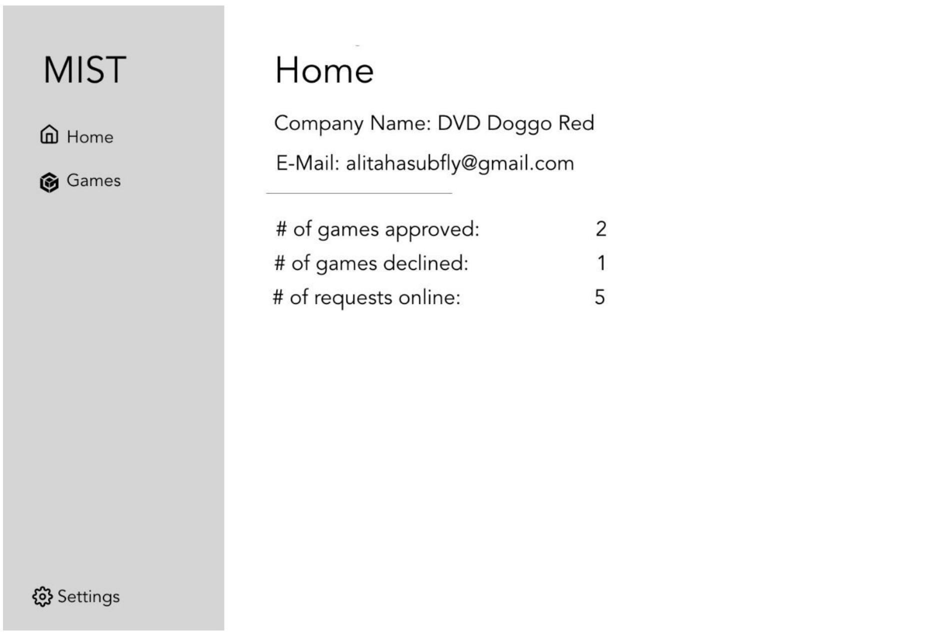
The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Addition to user SQL queries, the home page for the curator contains the number of followers and reviews left.

```
SELECT COUNT(*) AS num_of_followers

FROM followed_by f

WHERE f.c_ID = @a_ID;



SELECT COUNT(*) AS num_of_reviews_left

FROM review r

WHERE r.a_ID = @a_ID;
```

### 5.3.3 Developer Company

**User Interface:**



Home

Company Name: DVD Doggo Red

E-Mail: alitahasubfly@gmail.com

```
# of games approved:        2
# of games declined:        1
# of requests online:       5
```

*Figure 8: Home Page for Developer Company*

For a developer company, the user interface does not contain comments, reviews, wallet or friends information as developer companies do not have these functionalities. Instead, a number of developer's games approved, declined and current requests are shown.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Home page for the developer company contains the company's information, namely its name and email.

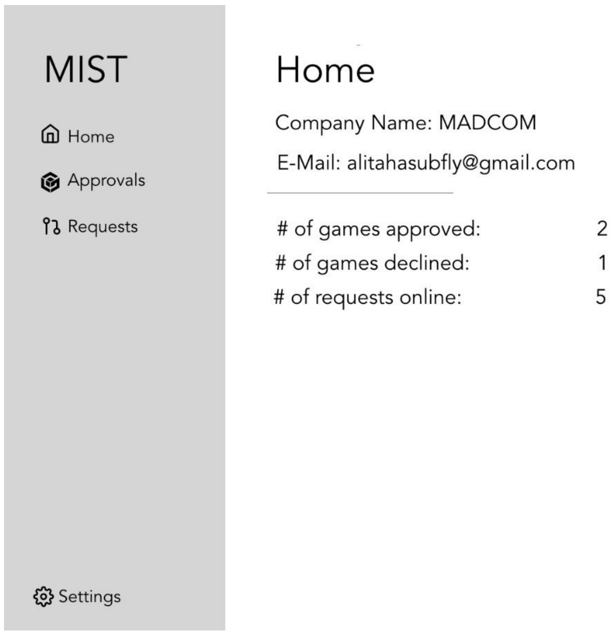
```
SELECT c_name, email_address

FROM Developer_Company  NATURAL JOIN Account

WHERE a_ID = @a_ID;
```

➔ Home page for the  developer company contains the number of games approved, games declined, number of requests online.

```
SELECT COUNT(*) AS num_of_games_approved

FROM takes t

WHERE t.a_ID = @a_ID AND t.state = “Approved”;


SELECT COUNT(*) AS num_of_games_declined

FROM takes t

WHERE t.a_ID = @a_ID AND t.tate = “Declined”;


SELECT COUNT(*) AS num_of_request_online

FROM takes t

WHERE t.a_ID = @a_ID AND (t.state <> “Declined” OR t.state <> “Approved”);
```

### *5.3.4 Publisher Company*

**User Interface:**



Home

Company Name: MADCOM

E-Mail: alitahasubfly@gmail.com

```
# of games approved:        2
# of games declined:        1
# of requests online:       5
```

*Figure 9: Home Page for Publisher Company*

For a publisher company, the user interface for the home page is similar to Developer Company as shown in *Figure 9*. However, the number of approved and declined games and requests are the publisher's decisions.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Home page for the developer company contains the number of games approved, games declined, number of requests online.

```
SELECT COUNT(*) AS num_of_games_approved

FROM asks a

WHERE a.a_ID = @a_ID AND a.state = "Approved";



SELECT COUNT(*) AS num_of_games_declined
```

```
FROM asks a
WHERE a.a_ID = @a_ID AND a.state = “Declined”;


SELECT COUNT(*) AS num_of_request_online
FROM asks a, takes t
WHERE t.r_ID = a.r_ID AND a.a_ID = @a_ID AND (a.state <> “Declined” OR a.state <>
“Approved”);
```

## 5.4 Library

**User Interface:**

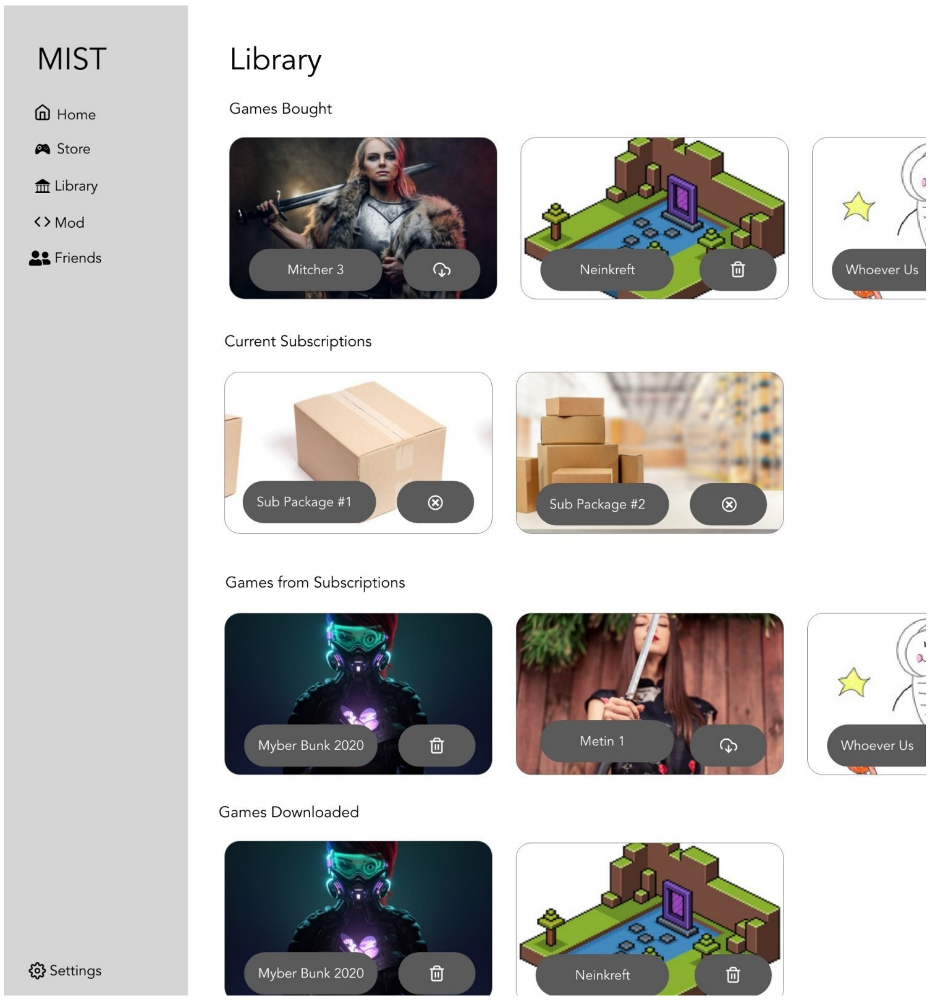


*Figure 10: Library Page for User and Curator*

Library page shows the games bought, downloaded and came from subscription packages. Additionally, subscribed subscription packages are shown as in *Figure 10*.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Display video games which the User bought.

```
SELECT v.g_name, v.g_image

FROM buys b, Video_Game v

WHERE b.g_ID=v.g_ID AND b.g_ID IN ( SELECT a.g_ID

FROM  Approved_Games a);
```

➔ Display video games which the User downloaded.

```
SELECT g.g_name, g.g_image

FROM install i, Approved_Games g

WHERE i.a_ID = @a_ID AND i.g_ID = g.g_ID;
```

➔ Display video games from subscriptions the User subscribes.

```
SELECT g.g_name, g.g_image

FROM contains c, subscribes s, Video_Game g

WHERE c.package_ID=s.package_ID AND s.a_ID = @a_ID

          AND g.g_ID = c.g_ID;
```

➔ Display subscriptions package(s) the User subscribes.

```
SELECT sp.package_name

FROM subscribes s, Subscription_Package sp

WHERE s.a_ID = @a_ID, sp.package_ID = s.package_ID;
```

➔ For distinguishing the download and uninstall buttons according to whether the user just bought the game or the user also downloaded the game.

```
SELECT b.g_name, b.g_image

FROM buys b, Approved_Games g

WHERE i.a_ID = @a_ID AND i.g_ID = g.g_ID;

MINUS

SELECT g.g_name, g.g_image

FROM install i, Approved_Games g

WHERE i.a_ID = @a_ID AND i.g_ID = g.g_ID;
```

➔ If the download button is clicked.

```
INSERT INTO install

VALUES (@a_ID, @g_ID, (SELECT g.version

                       FROM Approved_Games g

                       WHERE g.g_ID = @g_ID));
```

➔ If the uninstall button is clicked.

```
DELETE FROM install

WHERE a_ID=@a_ID AND g_ID = @g_ID;
```

➔ If the unsubscribe button is clicked.

```
DELETE FROM subscribes

WHERE a_ID=@a_ID AND package_ID = @package_ID;
```

## 5.5 Store Page

**User Interface for Store:**

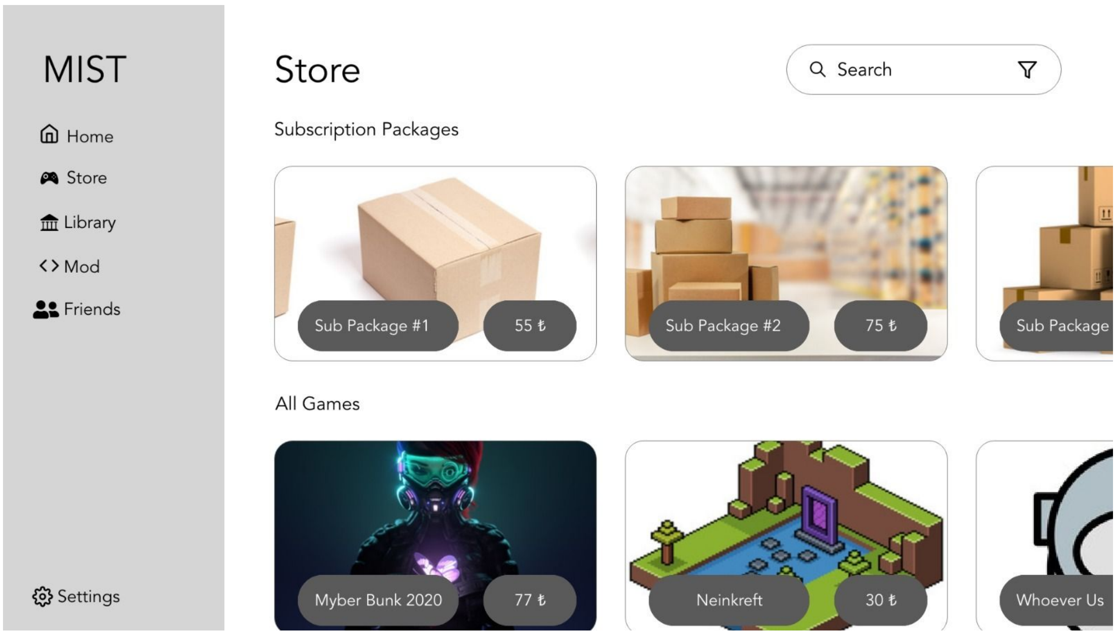


*Figure 11: Store Page for User and Curator*

Store page shows subscription packages and all games ordered by their rates. In store the name and the price of packages and games are displayed along with images.

**Corresponding SQL Statements:**

➔ In the Store Page, subscription packages are displayed.

```
SELECT package_name, price

FROM Subscription_Package;
```

➔ In the Store Page, video games are displayed by descending order of their rate.

```
WITH rate_of_games(g_ID, avg_rate) AS(

SELECT  g_ID, AVG(value) AS avg_rate

FROM rates

GROUP BY g_ID

ORDER BY avg_rate DESC)
```

```
SELECT a.g_name, a.g_price, a.g_image
FROM rate_of_games r, Approved_Games a
WHERE r.g_ID = a.g_ID;
```

**User Interface for Search-Filter-Sort:**

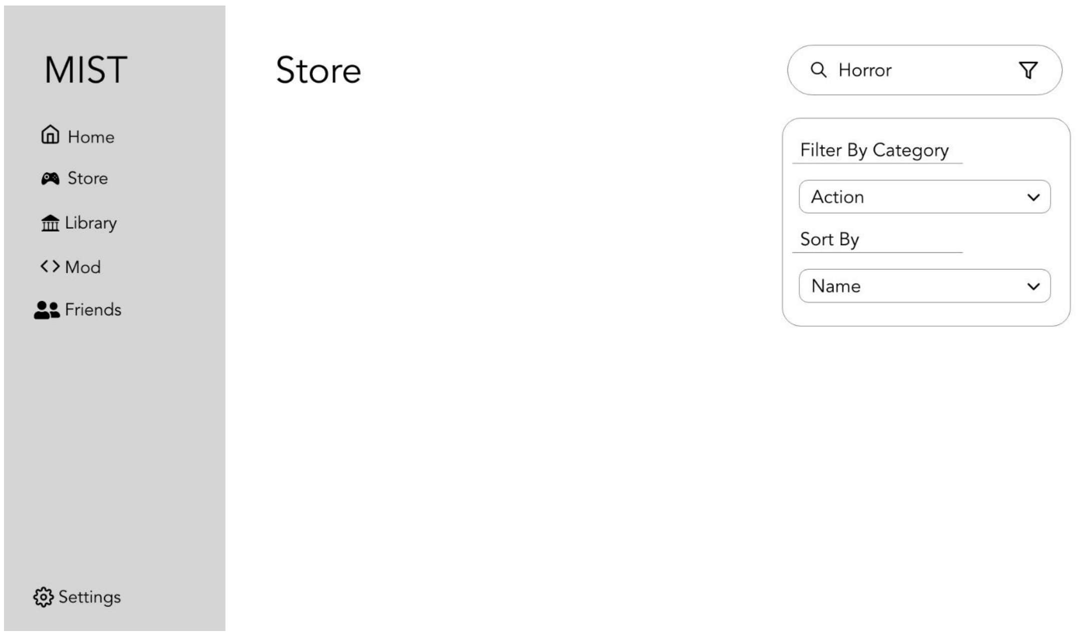


*Figure 12: Search-Filter-Store Page*

From the store page, users can search a word to find games containing that word in their games or descriptions. By clicking the button at the right side of the search field, as shown in *Figure 12*, users can filter games by their categories and sort games according to their name, price, rate and popularity (number of downloads).

**Corresponding SQL Statements:**

The value @key refers to the search query written by the user in the search field and @category_name refers to the filter option selected. Both are taken from the fronted.

➔ Filtering by a related word is used for the user to search games containing the word in either the game name or the game description.

```
SELECT a._name, a.g_price, a.g_image

FROM Approved_Games a

WHERE a.g_name like ‘%@key%’ OR a.g_description like ‘%@key%’ ;
```

➔ Filtering by categories is used for the user to search games.

```
WITH filter_by_category_and_name(a.g_name, a.g_price, a.g_image,
a.g_description ) AS(

SELECT a.g_name, a.g_price, a.g_image

FROM Approved_Games a

WHERE a.genre = @category_name,  a.g_name like ‘%@key%’ OR a.g_description
like ‘%@key%’; );

SELECT filter.g_name, filter.g_price, filter.g_image

FROM filter_by_category_and_name filter;
```

➔ Sorting by name can be chosen from the drop down section of sort by field.

```
SELECT a.g_name, a.g_price, a.g_image

FROM Approved_Games a

ORDER BY a.g_name ASC;



SELECT a.g_name, a.g_price, a.g_image

FROM Approved_Games a

ORDER BY a.g_name DESC;
```

➔ Sorting by ascending price can be chosen from the drop down section of sort by field.

```
SELECT a.g_name, a.g_price, a.g_image

FROM Approved_Games a

ORDER BY a.price ASC;
```

➔ Sorting by descending price can be chosen from the drop down section of sort by field.

```
SELECT a.g_name, a.g_price, a.g_image

FROM Approved_Games a

ORDER BY a.price DESC;
```

➔ Sorting by rate can be chosen from the drop down section of sort by field.

```
WITH rate_of_games(g_ID, avg_rate) AS(

SELECT  g_ID, AVG(value) AS avg_rate

FROM rates

GROUP BY g_ID

ORDER BY avg_rate ASC)

SELECT a.g_name, a.g_price, a.g_image

FROM rate_of_games r, Approved_Games a

WHERE r.g_ID = a.g_ID;
```

➔ Sorting by popularity can be chosen from the drop down section of sort by field.

```
WITH download_num_games(g_ID, total_downloads) AS(

SELECT  g_ID, COUNT(value) AS total_downloads

FROM buys
GROUP BY g_ID

ORDER BY total_downloads DESC)

SELECT t.g_name, t.g_price, t.g_image

FROM download_num_games t, Approved_Games a

WHERE r.g_ID = t.g_ID;
```

## 5.6 Wallet Options Page

### User Interface Add Credit Card:

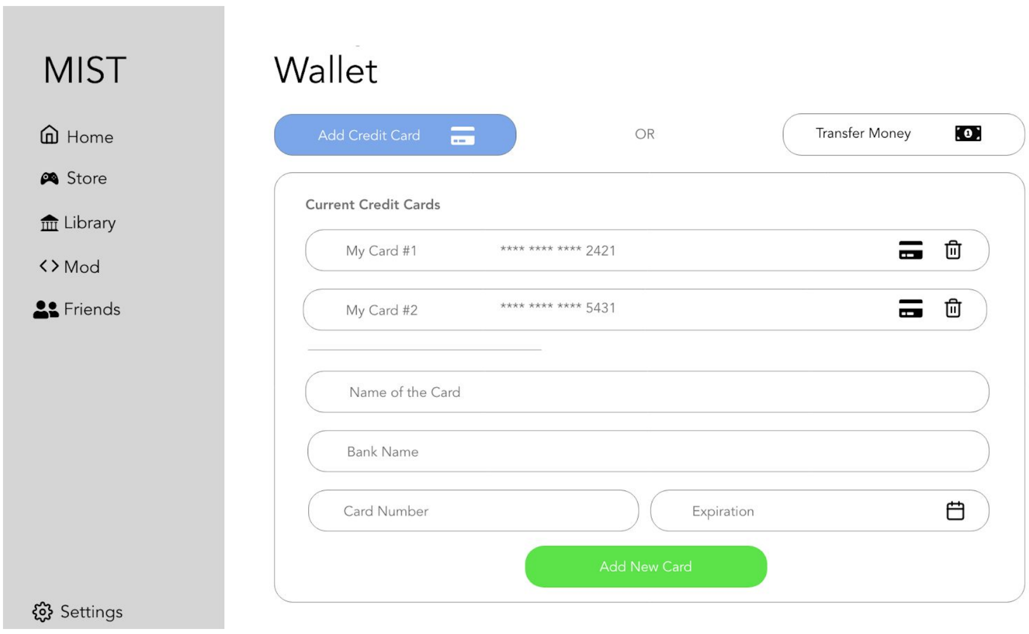


*Figure 13: Wallet Options Page for Add Credit Card Tab*

After clicking wallet options, the wallet page is displayed as in *Figure 13*. Users can add credit cards for transferring money to the wallet. Current credit cards and fields for new credit cards are displayed.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The other values @card_number refers to card number, @bank_name refers to bank name, @name_of_the_card refers to the name of the card and @expiration refers to expiration date which are all taken from the frontend. The value @w_ID refers to the wallet of the corresponding user and stored during the site is open.

➔ Show current cards the user has.

```
SELECT c.card_ID, c.name

FROM Wallet w, include i, Credit_Card c

WHERE w.a_ID=@a_ID AND i.w_ID=w.w_ID AND c.card_ID=i.card_ID;
```

➔ Add a credit card to the wallet.

```
INSERT INTO Credit_Card

VALUES (@card_number, @bank_name, @name_of_the_card, @expiration);



INSERT INTO include

VALUES (@card_number, @w_ID, @a_ID);
```

**User Interface Transfer Money:**

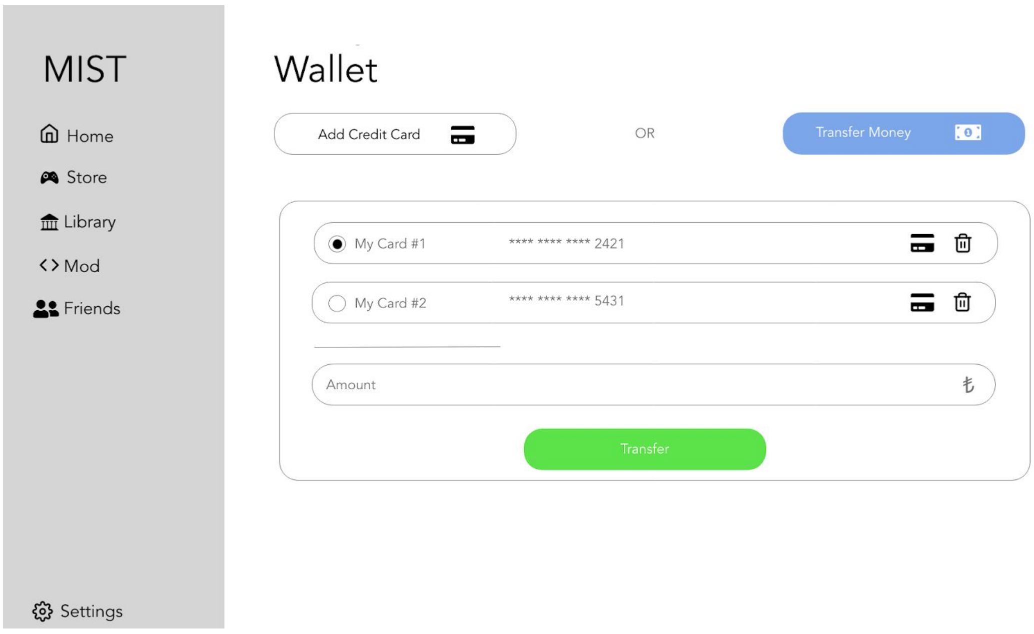


*Figure 14: Wallet Options Page for Transfer Money Tab*

After choosing the transfer money tab, the page is displayed in *Figure 14*. User chooses a credit card and enters the amount of money to transfer money to the wallet.

### Corresponding SQL Statements:

@Amount refers to the amount the user has entered. @card_ID refers to the card id that the user has selected. The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

→ Transfer money to wallet and update balance.

```
UPDATE Wallet w

SET balance = balance + @Amount

WHERE w.a_ID = @a_ID;
```

→ Deleting the credit card on delete button pressed

```
DELETE FROM Credit_Card c

WHERE  c.card_ID=@card_ID;
```

### User Interface Payment:

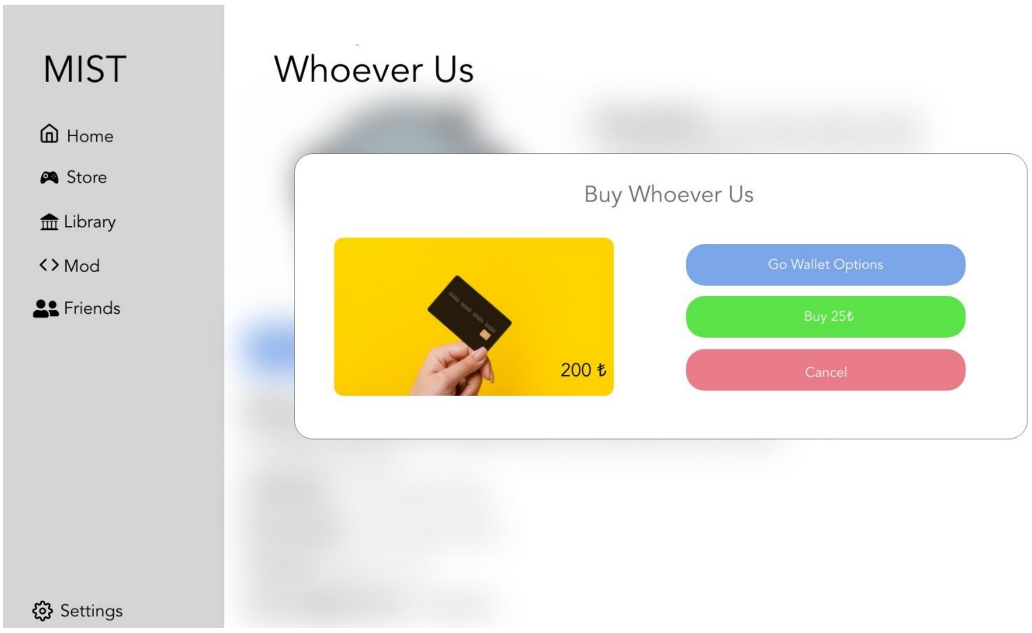


*Figure 15: Payment Page for Games and Subscriptions*

After clicking buy or subscribe from games and subscription packages, the pop-up is displayed for the payment. If the user decides to buy, wallet balance is decreased and the game or the subscription package is added to the user's library. *Figure 15* shows the pop-up page after clicking the buy button from a video game.

### Corresponding SQL Statements:

@current_date refers to the current date taken from the system and @package_ID refers to the chosen package ID. The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Update wallet amount after purchasing game or subscription package.

```
UPDATE Wallet W

SET balance = balance - (SELECT g_price

                          FROM Approved_Games

                         WHERE g_ID = @g_ID)

WHERE W.a_ID = @a_ID;


UPDATE Wallet W

SET balance = balance - (SELECT price

                          FROM Subscription_Package

                         WHERE package_ID = @package_ID)

WHERE W.a_ID = @a_ID;
```

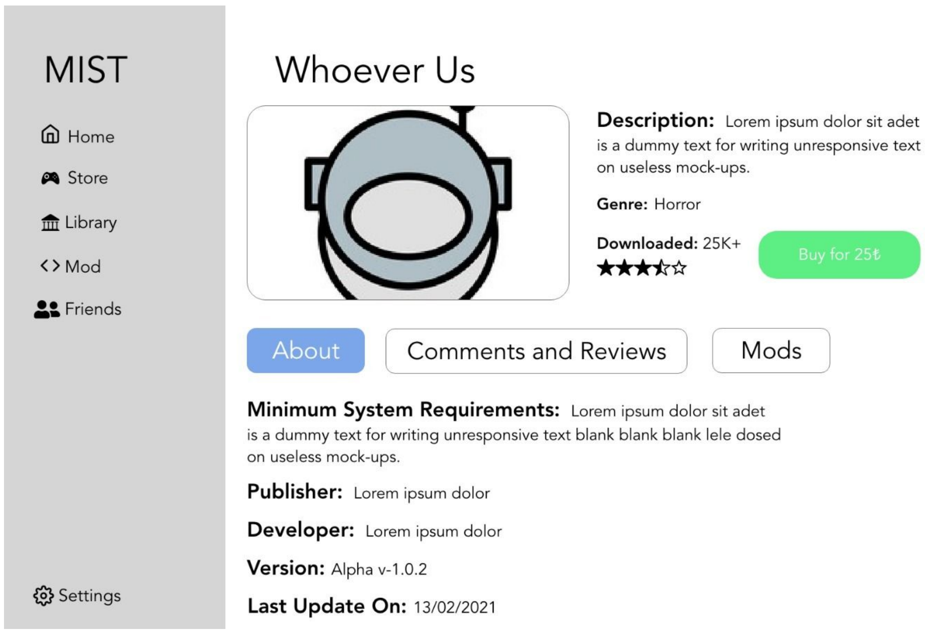
➔ Add the game or the subscription package to relations.

```
INSERT INTO buys

VALUES (@a_ID, @g_ID, @current_date);

INSERT INTO subscribes

VALUES (@a_ID, @package_ID, @current_date);
```

## 5.7 Video Game

### *5.7.1 User*

**User Interface About Tab:**



Whoever Us

**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Genre:** Horror

**Downloaded:** 25K+

★★★⯪☆

**Minimum System Requirements:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text blank blank blank lele dosed on useless mock-ups.

**Publisher:** Lorem ipsum dolor

**Developer:** Lorem ipsum dolor

**Version:** Alpha v-1.0.2

**Last Update On:** 13/02/2021

*Figure 16: Video Game Page About Tab*

Video game page shows information about a video game. In About Tab, minimum system requirements, publisher and developer of the game, current version and the last update is shown. If the game is newly published and the developer company did not update the game, the last update shows the publish date.

**Corresponding SQL Statements:**

The value @g_ID refers to the game id that is shown in the page.

➔ The information about video game:

```
SELECT g_name, g_description, g_version, g_image, g_price, genre,
g_requirements

FROM Approved_Games

WHERE g_ID = @g_ID;
```

➔ The information about last update date, publisher and developer of the game :

```
SELECT c_name AS devoloper_name

FROM develops d, Company C

WHERE d.g_ID = @g_ID AND d.a_ID = C.a_ID;


SELECT c_name AS publisher_name

FROM publish p, Company C

WHERE p.g_ID = @g_ID AND p.a_ID = C.a_ID;
```

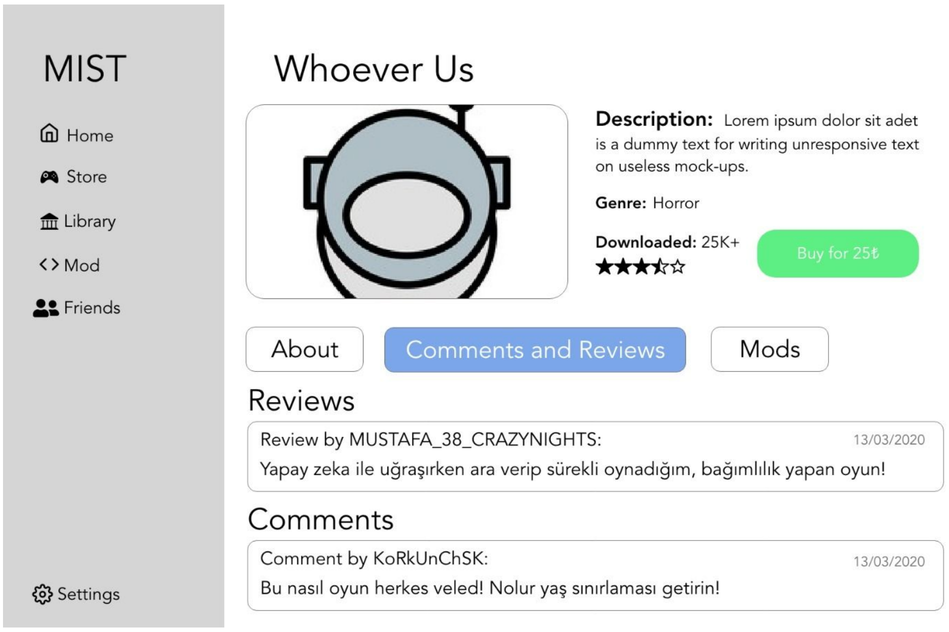
➔ The information about the last update:

```
SELECT date

FROM update u, Video_Game vg

WHERE u.g_ID = vg.g_ID AND vg.g_ID = @g_ID;
```

➔ The information about the total number of downloads:

```
SELECT COUNT(*) AS num_of_download

FROM buys

WHERE g_ID = @g_ID;
```

## User Interface Comments and Reviews Tab:



**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Genre:** Horror

**Downloaded:** 25K+

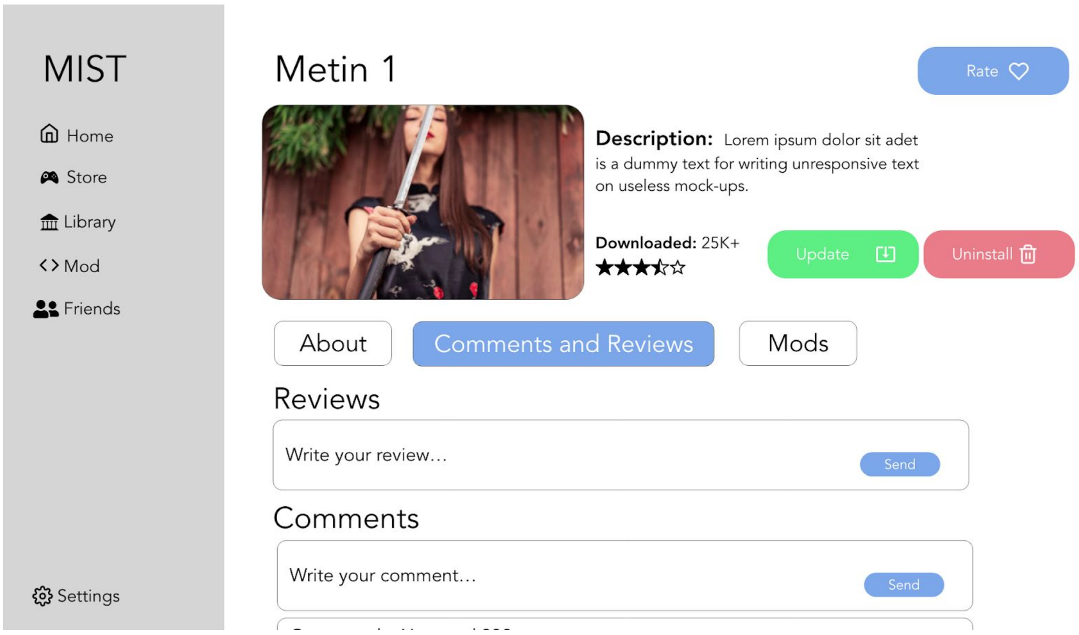
Reviews

Review by MUSTAFA_38_CRAZYNIGHTS: 13/03/2020
Yapay zeka ile uğraşırken ara verip sürekli oynadığım, bağımlılık yapan oyun!

Comments

Comment by KoRkUnChSK: 13/03/2020
Bu nasıl oyun herkes veled! Nolur yaş sınırlaması getirin!

*Figure 17: Video Game Page Comments and Reviews Tab*



**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Downloaded:** 25K+

*Figure 18: Video Game Page Comments and Reviews Tab for Adding Comments and Reviews*

In Comments and Reviews Tab, reviews and comments that have been made to the video game are shown. In Figure 17, as the user has not bought the game yet, comments or reviews cannot be entered in this case. If the user has installed the game, then the user is able to write a comment to the game. In addition to that, curators are able to write a review of the game if they have installed the game. In order to display the full page structure, curators view the page of comments and review tab structure including writing a review and comment is shown in Figure 18.

**Corresponding SQL Statements:**

The value @g_ID refers to the game id that is shown in the page. The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The values @c_date refers to the date that user is sending the comment and taken from the system, @c_description refers to the comment written by the user and taken from the frontend. The values @date refers to the date of the review that is being written and will be taken from the system, @text refers to the review written by the curator and will be taken from the frontend..

➔ Display Reviews:

```
SELECT r.date, r.text, u.u_name

FROM review r, Curator c, Approved_Games g, User u

WHERE u.a_ID = r.a_ID AND u.a_ID = c.a_ID AND g.g_ID = r.g_ID AND
g.g_ID = @g_ID;
```

➔ Display Comments:

```
SELECT c.date, c.text, u.u_name

FROM comments_on c, User u

WHERE c.a_ID = u.a_ID AND c.g_ID = @g_ID
```

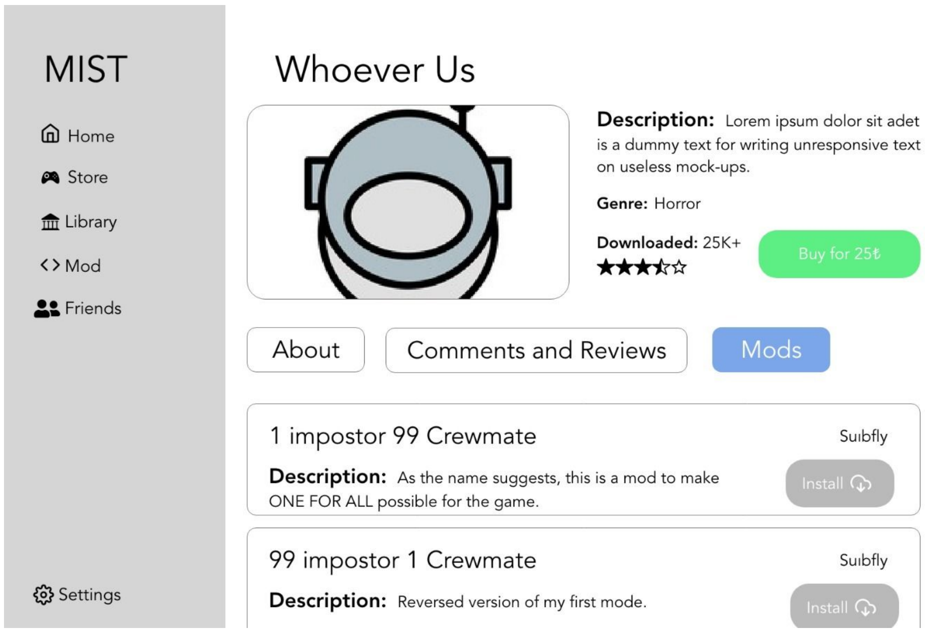
➔ Add Reviews (For curator):

```
INSERT INTO review

VALUES (@a_ID, @text, @date);
```

➔ Add Comment (For User):

```
INSERT INTO comments_on(a_ID, g_ID, date, text)

VALUES (@a_ID, @g_ID, @date, @text)
```

**User Interface Mods Tab:**



**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Genre:** Horror

**Downloaded:** 25K+

★★★⯪☆

1 impostor 99 Crewmate — Suıbfly

**Description:** As the name suggests, this is a mod to make ONE FOR ALL possible for the game.

99 impostor 1 Crewmate — Suıbfly

**Description:** Reversed version of my first mode.

*Figure 19: Video Game Page Mods Tab*

In the Mods tab, built mods for the game is shown. Because the user had not bought and downloaded the game yet, mods cannot be installed, therefore, install buttons are not available for the case in *Figure 19*.

**Corresponding SQL Statements:**

The value @g_ID refers to the game id that is shown in the page.

➔ Display mods of the current game:

```
SELECT M.m_name, M.m_description, U.u_name

FROM Mod M, for_m f, builds b, User U

WHERE f.g_ID = @g_ID AND M.m_ID = b.m_ID AND U.a_ID = b.a_ID AND f.m_ID =
M.m_ID;
```

## User Interfaces for Buying, Installing, Returning, Uninstalling and Updating

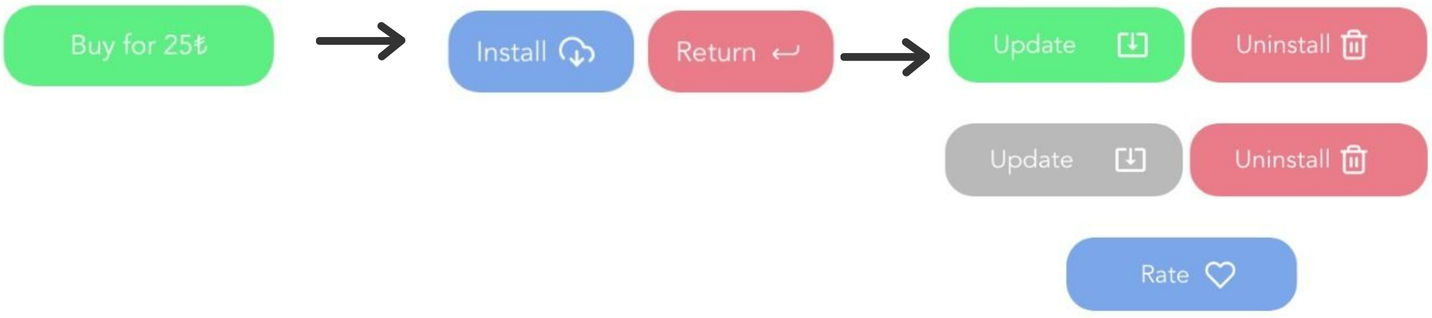
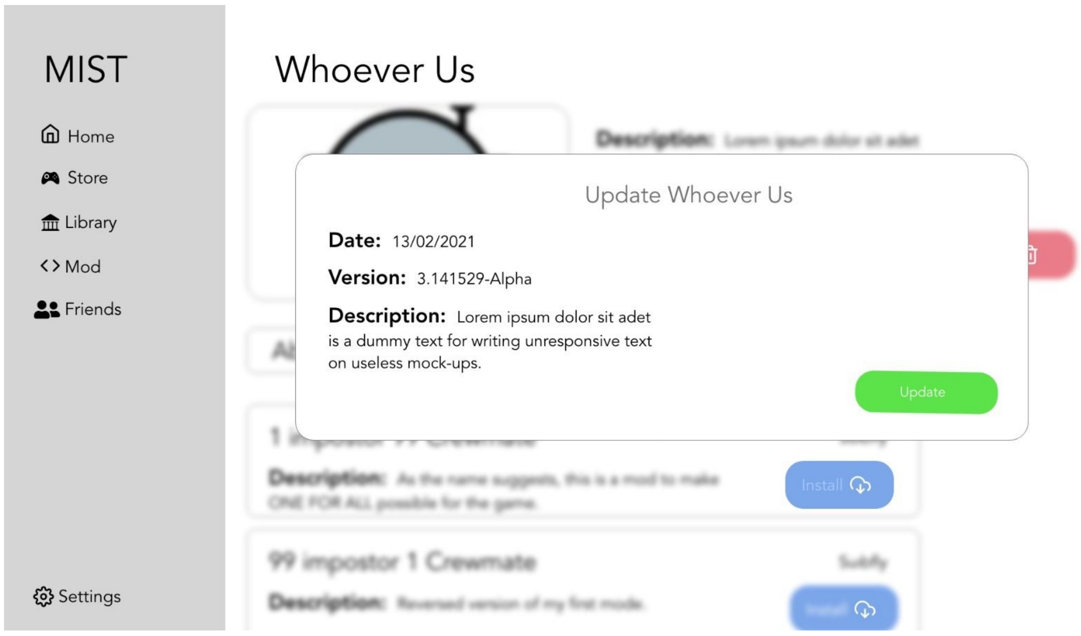


*Figure 20: Flow of User Operations on Video Game*



Update Whoever Us

**Date:** 13/02/2021

**Version:** 3.141529-Alpha

**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

*Figure 21: Pop-Up For Update Information*

1. Before buying the game, a user cannot install, return, update or uninstall the game.
2. After buying the game, the user can return the game or install the game.
3. After installing the game, if any update is done by the developer company, the user can update the game seeing information about update attributes as in Figure 21. Additionally, the user can leave comments or if curator, enter reviews. All users can rate the game after downloading the game.The user can also uninstall the game and go back to the second stage of *Figure 20*.

**Corresponding SQL Statements:**

The value @g_ID refers to the game id that is shown in the page. The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @value refers to the rate that is given by the curators and users to a game and taken from the frontend.

- ➔ Insertion of Bought games can be reached from section **4.6**, payment page.
- ➔ Download the game.

```
INSERT INTO install

VALUES (@a_ID, @g_ID, (SELECT g.version

                       FROM Approved_Games g

                       WHERE g.g_ID = @g_ID));
```

➔ Return the game

```
DELETE FROM buys

WHERE buys.a_ID = @a_ID AND buys.g_ID = @g_ID;
```

➔ Display the update

```
SELECT description, version_no, date

FROM updates

WHERE g_ID = @g_ID;
```

➔ Update version number of Users downloads

```
SELECT vers_no

FROM Video_Game

WHERE g_ID = @g_ID

UPDATE install
```

```
SET version_no = vers_no

WHERE a_ID = @a_ID AND g_ID = @g_ID;
```

➔ Rate the game

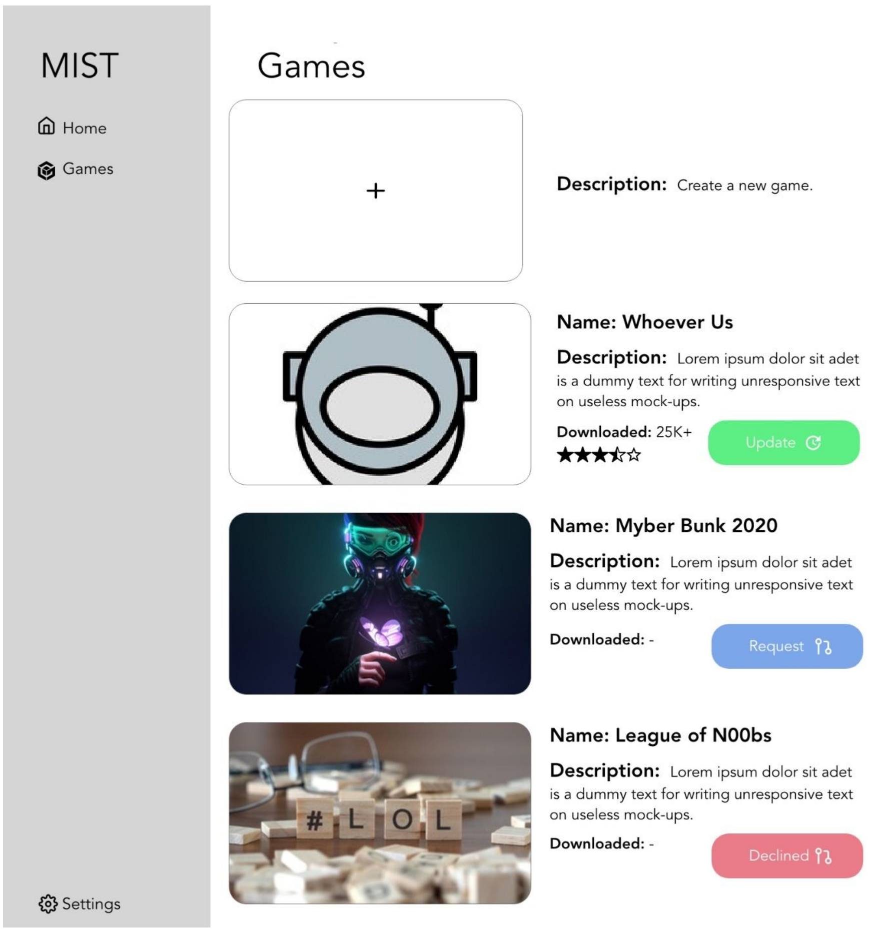
```
INSERT INTO rates

VALUES (@a_ID, @g_ID, @value);
```

➔ Uninstall the game

```
DELETE FROM install

WHERE a_ID = @a_ID AND g_ID = @g_ID;
```

## 5.8 Games Page for Developer Company

**User Interface Displaying Games:**



**Description:** Create a new game.

**Name: Whoever Us**
**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.
**Downloaded:** 25K+
★★★⯪☆

**Name: Myber Bunk 2020**
**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.
**Downloaded:** -

**Name: League of N00bs**
**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.
**Downloaded:** -

*Figure 22: Games Page for Developer Company*

In the games page of developer company user type, there are several options that a developer company can do.

1. A developer company can create games by clicking the plus icon in the beginning of the page.
2. A developer company can update their games that are published on the store.
3. A developer company can send a publish request to any publisher company by clicking the blue request button.
4. A developer company can send a new publish request for a game that has been declined by the selected company. Hence, by this, the developer company can both see the declined status from the button and send a new publish request to a publisher company.

## Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ Display Developer Company Approved Games

```
SELECT vg.g_name, vg.g_description, vg.g_image, COUNT(i.a_ID), AVG(r.value)
FROM develops d, about a, takes t, Video_Game vg, install i, rates r, asks ask,
      request req
WHERE t.state = “Approved” AND vg.g_ID=a.g_ID AND a.r_ID = req.r_ID AND
        t.r_ID = req.r_ID AND ask.r_ID = t.r_ID AND ask.a_ID=@a_ID AND
        d.a_ID=@a_ID AND d.g_ID = vg.g_ID AND i.g_ID = d.g_ID
        AND r.g_ID = d.g_ID
GROUP BY g_ID;
```

➔ Display Developer Company Games waiting in request.

```
SELECT vg.g_name, vg.g_description, vg.g_image
```

```
FROM develops d, about a, takes t, Video_Game vg, asks ask, Request req
WHERE (t.state <> “Approved” OR  t.state < > “Declined” AND vg.g_ID=a.g_ID
        AND a.r_ID = req.r_ID AND t.r_ID = req.r_ID AND ask.r_ID = t.r_ID
        AND ask.a_ID=@a_ID AND d.a_ID=@a_ID AND d.g_ID = vg.g_ID
MINUS
SELECT vg.g_name, vg.g_description, vg.g_image
FROM Video_Game vg, publish p
WHERE vg.g_ID = publish.g_ID;
```

➔ Display Developer Company declined Games.

```
SELECT vg.g_name, vg.g_description, vg.g_image
FROM develops d, about a, takes t, Video_Game vg, asks ask, Request req
WHERE (t.state = “Declined”) AND vg.g_ID=a.g_ID
        AND a.r_ID = req.r_ID AND t.r_ID = req.r_ID AND ask.r_ID = t.r_ID
        AND ask.a_ID=@a_ID AND d.a_ID=@a_ID AND d.g_ID = vg.g_ID;
```

**User Interface Creating a Game:**

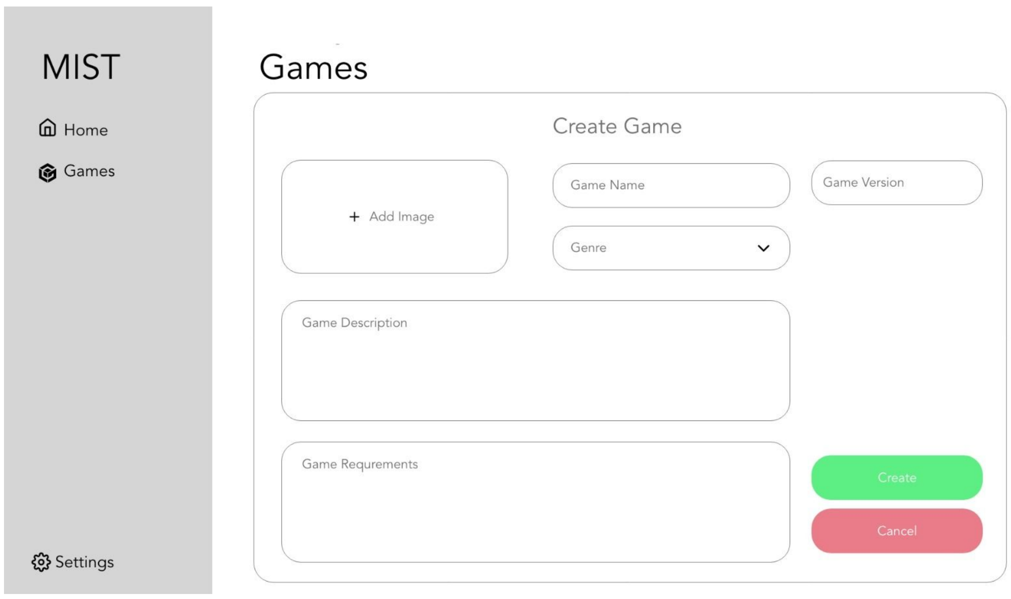


*Figure 23: Create a Game Screen When Create a Game Selected*

When the developer company clicks a new game field, the page will be shown as in *Figure 23*. The developer company will add an image for the game, enter name, genre and version information along with the description and minimum system requirements.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The values @g_name refers to game name, @g_version refers to game version, @g_description refer to game description, @g_image refers to image of the game, @genre refers to genre of the game and @g_requirements refers to requirements of the game and all taken from the frontend during creating a game.

➔ Create a new game and add data to corresponding tables.

```
INSERT INTO Video_Game(g_name, g_version, g_description, g_image, g_price, genre,
g_requirements)

VALUES (@g_name, @g_version, @g_description, @g_image, 0, @genre, ,
@g_requirements);



INSERT INTO develops

VALUES (@a_ID, SELECT g_ID

                    FROM Video_Game

                    WHERE g_ID=(SELECT MAX(g_ID) FROM Video_Game));
```

### User Interface Updating a Game:

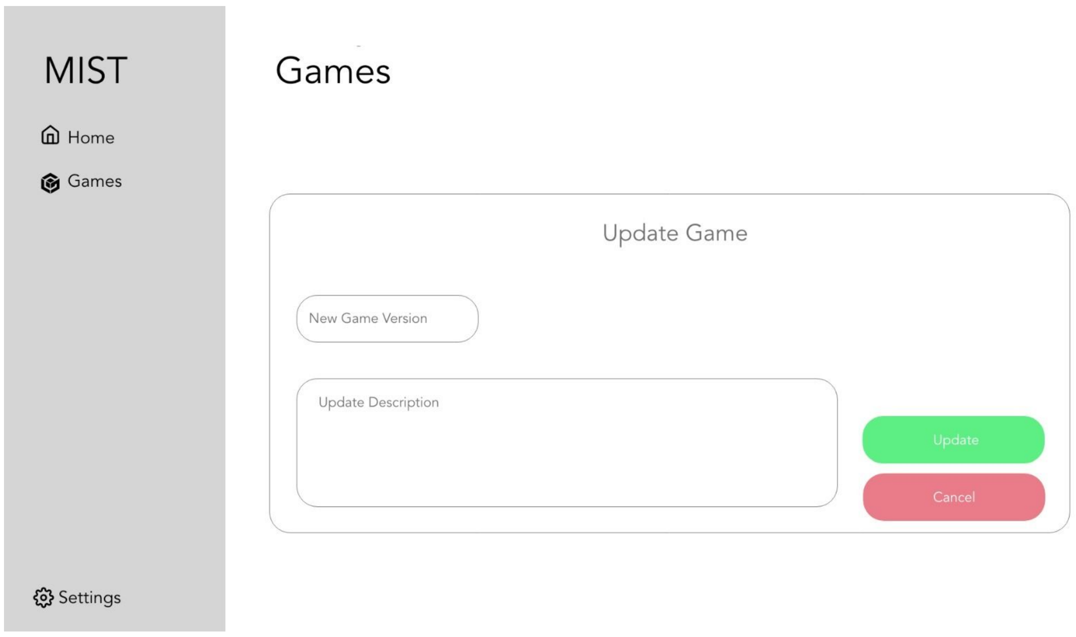


*Figure 24:Update the Game Screen When Update is Selected*

Developer companies can update their own game after clicking the Update button. A new screen will be shown as in *Figure 24*. The developer company will enter a new version and the update description. The date of update will be automatically obtained from the system.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @g_ID refers to the game id that is shown in the page. The values @g_version refers to the version of the game, @g_description refers to the description of the game and @date refers to the update date of the game. @date will be taken from the system and other values will be taken from the frontend.

→ Update attributes of a video game and add new data to corresponding relations.

```
UPDATE Video_Game

SET g_version = @g_version
```

```
WHERE g_ID = @g_ID;

INSERT INTO updates

VALUES (@g_ID, @a_ID, @g_version, @g_description, @date);
```

**User Interface Sending Request:**

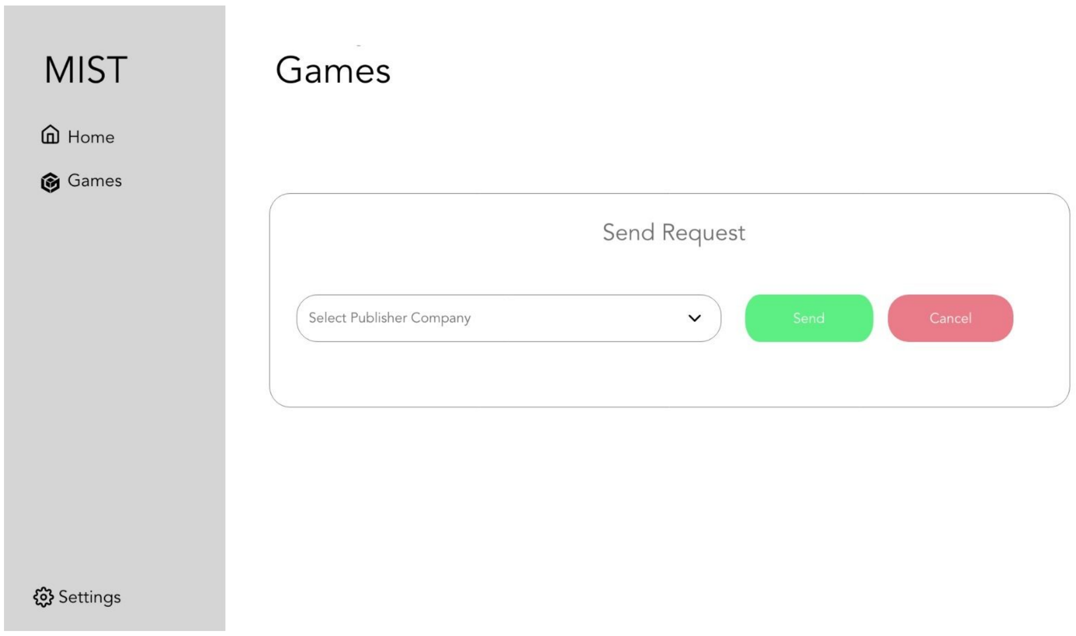


*Figure 25: Request Screen  for The Developer Company Games*

The developer company sends a request to publisher companies for publishing their game. The publisher company is chosen from the drop down field as can be seen in *Figure 25*.

**Corresponding SQL Statements:**

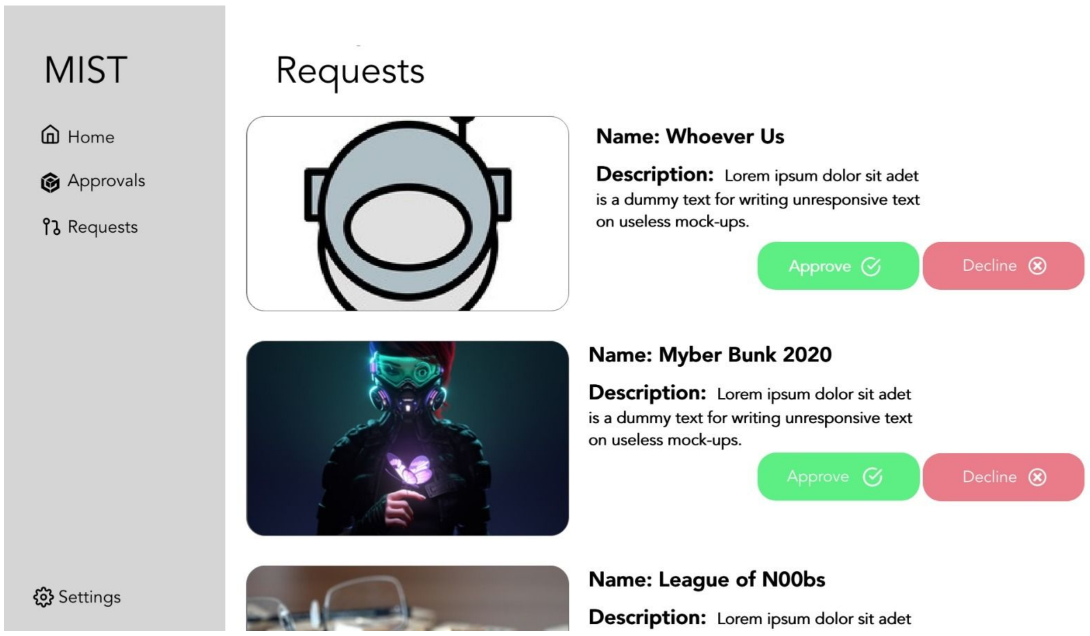
The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @g_ID refers to the game id that is shown in the page.

```
INSERT INTO Request()

VALUES ();


INSERT INTO asks
```

```
VALUES (SELECT r_ID
          FROM Request
          WHERE r_ID=(SELECT MAX(r_ID) FROM Request), @a_ID);


INSERT INTO takes
VALUES (SELECT r_ID
          FROM Request
          WHERE r_ID=(SELECT MAX(r_ID) FROM Request), @a_ID, ‘ ’);


INSERT INTO about
VALUES (SELECT r_ID
          FROM Request
          WHERE r_ID=(SELECT MAX(r_ID) FROM Request), @g_ID);
```

## 5.9 Request Page for Publisher Company

**User Interface:**



**Name: Whoever Us**

**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Name: Myber Bunk 2020**

**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

**Name: League of N0Obs**

**Description:** Lorem ipsum dolor sit adet

*Figure 26: Requests Page for the Publisher Company*

The publisher company type has a requests page that they can see the game requests came for them to publish. The publisher company either approves or declines the requests from this page as demonstrated in *Figure 26*.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @r_ID refers to the request id of a request and taken from the frontend.

➔ Display Publisher Company Requested Games

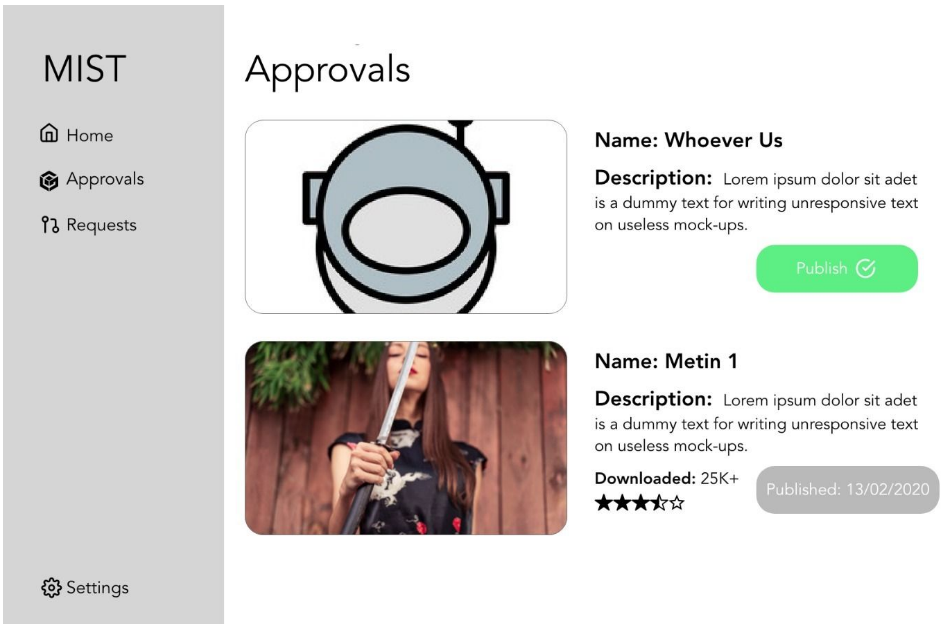
```
SELECT vg.g_name, vg.g_description, vg.g_image
FROM develops d, about a, takes t, Video_Game vg, asks ask, request req
WHERE (t.state <> “Approved” OR  t.state < > “Declined”) AND vg.g_ID=a.g_ID
        AND a.r_ID = req.r_ID AND t.r_ID = req.r_ID AND ask.r_ID = t.r_ID
        AND t.a_ID=@a_ID AND d.a_ID=@a_ID AND d.g_ID = vg.g_ID
MINUS
SELECT vg.g_name, vg.g_description, vg.g_image
FROM Video_Game vg, publish p
WHERE vg.g_ID = publish.g_ID AND p.a_ID = @a_ID;
```

➔ Whether the publisher company approves the games or not.

```
UPDATE takes
SET state=@decision
WHERE r_ID=@r_ID;
```

## 5.10 Approval Page for Publisher Company

### User Interface Approvals:



Name: Whoever Us

Description: Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

Name: Metin 1

Description: Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.

Downloaded: 25K+

*Figure 27: Approvals Page for the Publisher Company*

Publisher companies have an approval page to see the games they have approved before. From this page, they can see the information about the game and the status of whether they have published or not. If they have not published yet, the publish button is used.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

→ Display Publisher Company Approved but not Published Games

```
SELECT vg.g_name, vg.g_description, vg.g_image
FROM develops d, about a, takes t, Video_Game vg, asks ask, Request req
WHERE   t.state= “Approved” AND vg.g_ID=a.g_ID
        AND a.r_ID = req.r_ID AND t.r_ID = req.r_ID AND ask.r_ID = t.r_ID
        AND t.a_ID=@a_ID AND d.a_ID=@a_ID AND d.g_ID = vg.g_ID
```

```
MINUS

SELECT vg.g_name, vg.g_description, vg.g_image

FROM Video_Game vg, publish p

WHERE vg.g_ID = publish.g_ID AND p.a_ID = @a_ID;
```

➔ Display Published Games of the Publisher Company

```
SELECT vg.g_name, vg.g_description, vg.g_image

FROM Video_Game vg, publish p

WHERE vg.g_ID = publish.g_ID AND p.a_ID = @a_ID;
```

**User Interface Publishing Approvals Screen:**

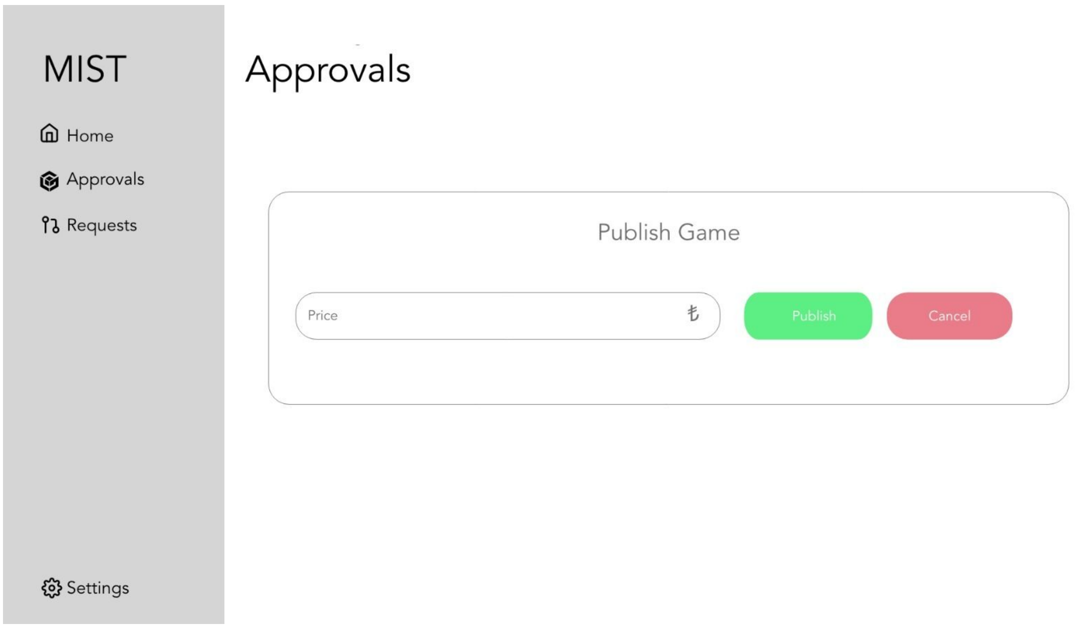


*Figure 28: Publish Screen*

After clicking the publish button from the Approvals page, the publisher company sets the price of the game and then publishes as in *Figure 28*.

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The values @chosen_amount refers to the price of the game, @selected_game_ID refers to the id of the game selected to be approved and published and @current_date refers to the date that the publication made. @current date will be taken from the system, other values will be taken from the frontend.

➔ Approval Page for Publisher Company

```
UPDATE  Video_Game(

SET(g_price =@chosen_amount )

WHERE g_ID = @selected_game_ID);



INSERT INTO publish

VALUES (@selected_game_ID, @a_ID, @current_date);
```

## 5.11 Subscription Package

**User Interface:**

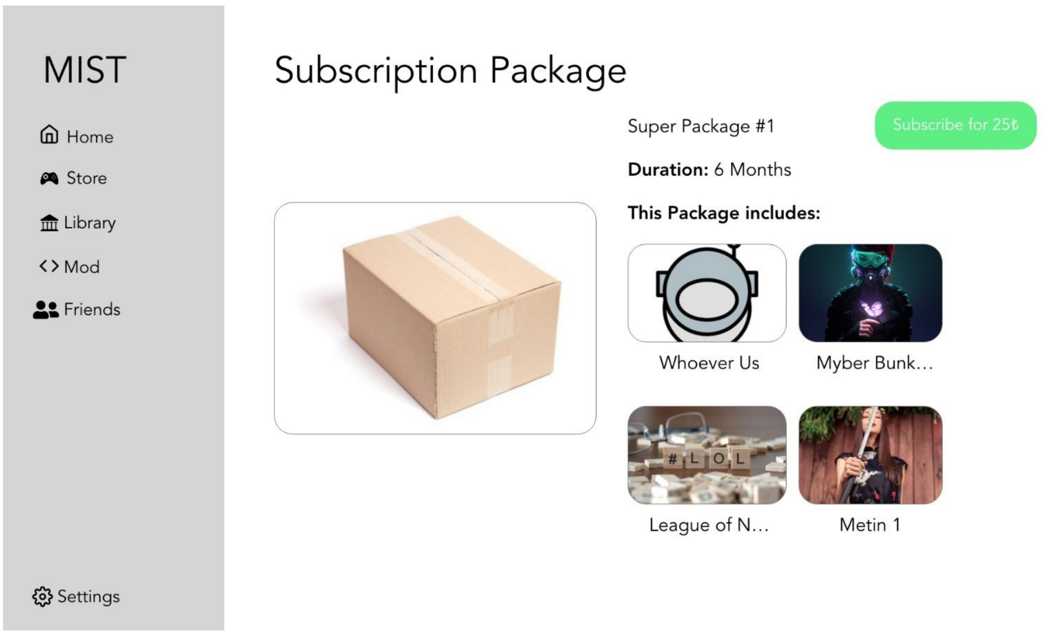


*Figure 28: Subscription Package Page*

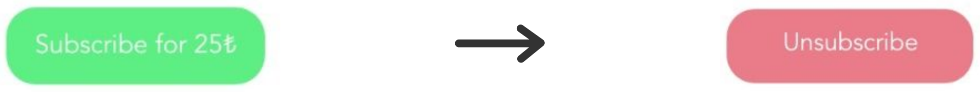


*Figure 30: Flow of Buttons for Different Cases*

In the subscription package page, users can see related information about the package and all games that the subscription package contains as shown in *Figure 29*. After subscribing, users can also unsubscribe.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @package_ID refers to the id of the subscription package that the user has chosen. The value @current_date refers to the date that the user has subscribed to the package. The @current_date value is taken from the system.

→ Display subscription package details

```
SELECT sp.package_name, sp.price, sp.duration

FROM Subscription_Package sp

WHERE sp.package_ID = @package_ID;



SELECT vg.g_image, vg.g_name

FROM Subscription_Package sp, contains c, Video_Game vg

WHERE sp.package_ID = @package_ID AND sp.package_ID = c.package_ID

          AND c.g_ID = vg.g_ID;
```

➔ Update subscribes table and wallet after subscribe button is pressed

```
INSERT INTO subscribes

VALUES (@a_ID, @package_ID, @current_date);



UPDATE Wallet W

SET balance = balance - (SELECT price

                          FROM Subscription_Package

                         WHERE package_ID = @package_ID)

WHERE W.a_ID = @a_ID;
```

➔ Update wallet after unsubscribe button is pressed

```
DELETE FROM subscribes

WHERE a_ID = @a_ID AND package_ID =@package_ID;



UPDATE Wallet W

SET balance = balance + (SELECT price

                          FROM Subscription_Package

                         WHERE package_ID = @package_ID)

WHERE W.a_ID = @a_ID;
```

## 5.12 Mod Page

### User Interface Mod Page:

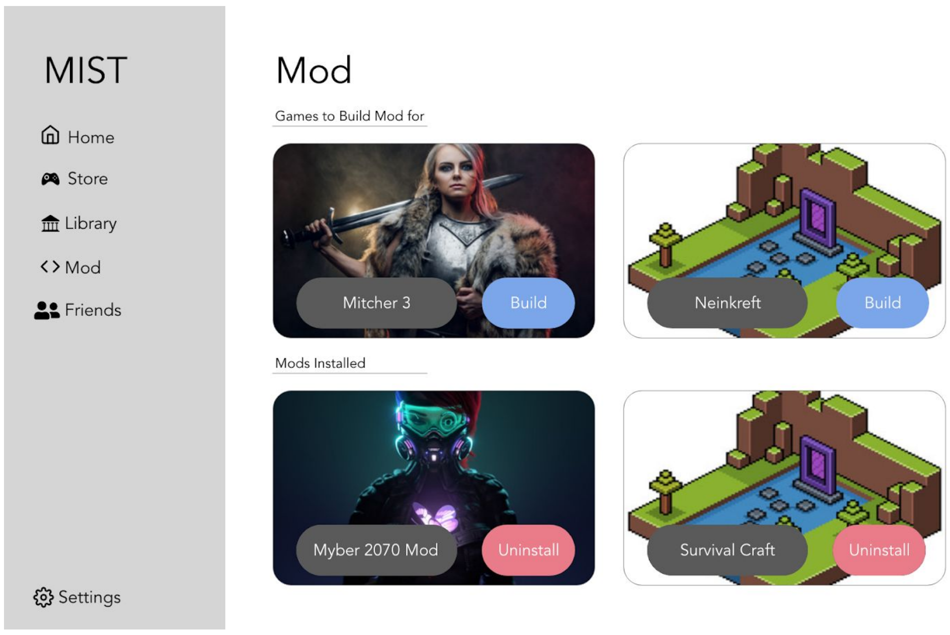


*Figure 31: Mod Page for Users*

In the Mod page, users can see their games that they can build a mod for and the mods they have installed before as in *Figure 31*.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

→ Display games to build mod for

```
SELECT g_name, g_image

FROM Approved_Games;



SELECT g_name, g_image

FROM install

WHERE a_ID = @a_ID;
```

**User Interface Create Mod Screen:**

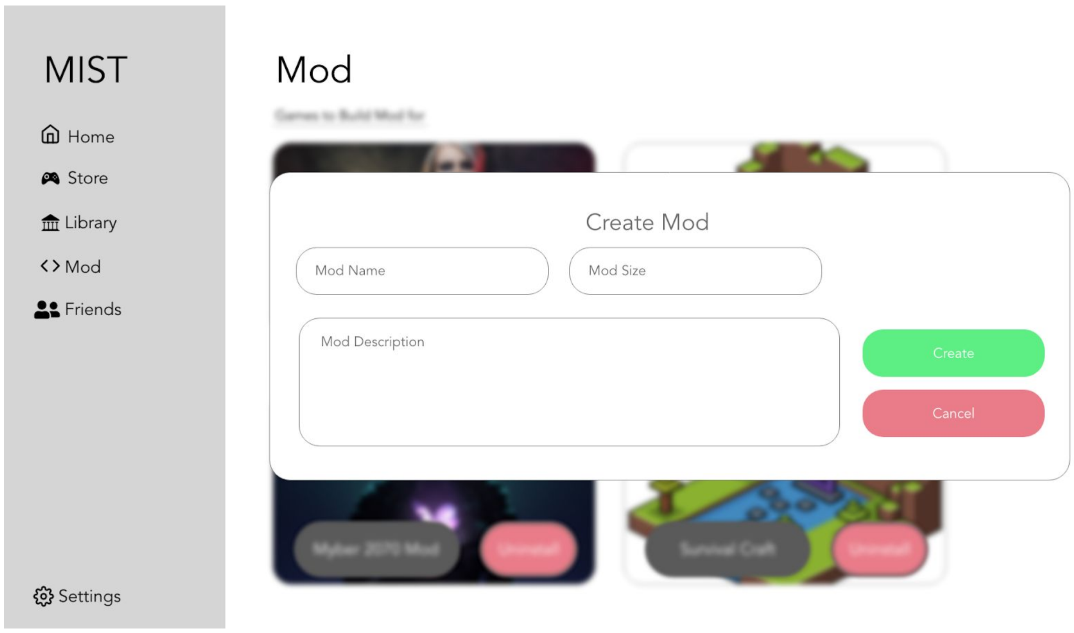


*Figure 32: Create Mod Screen*

When the build button is clicked from the Mod page, users enter the mod name and description along with its size in order to create a mod for the chosen game.

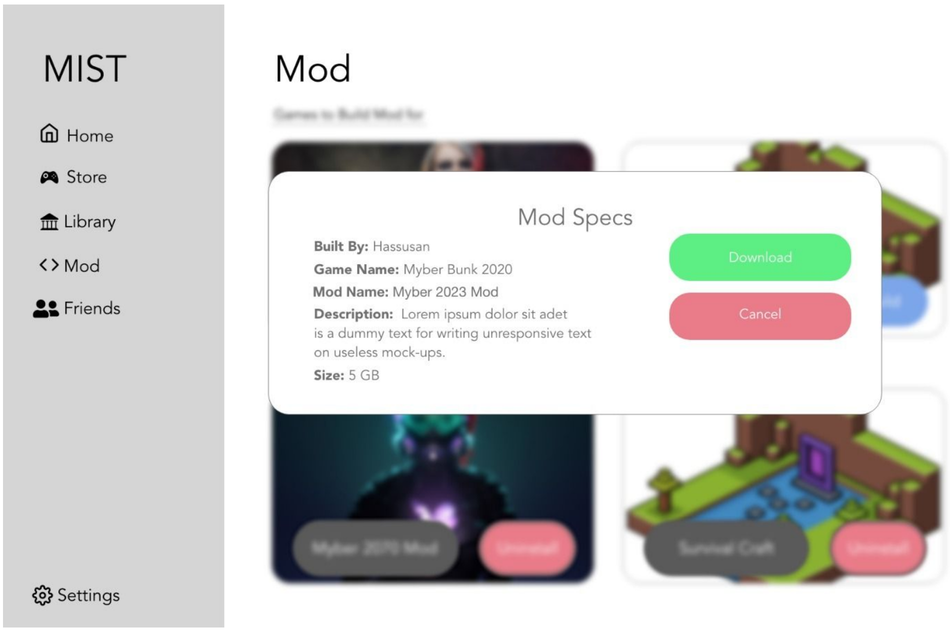
**Corresponding SQL Statements:**

The values @m_name refers to the name of the mod, @m_description refers to the description of the mod and @m_size refers to the size of the mod. All of the values will be taken from the frontend.

➔ Record new mod data to the table

```
INSERT INTO  Mod(m_name, m_description, m_size)

VALUES (@m_name, @m_description, @m_size);
```

### User Interface Mod Information Screen:



**Built By:** Hassusan
**Game Name:** Myber Bunk 2020
**Mod Name:** Myber 2023 Mod
**Description:** Lorem ipsum dolor sit adet is a dummy text for writing unresponsive text on useless mock-ups.
**Size:** 5 GB

*Figure 33: Mod Information Screen*

When a mod is clicked, the information about that mod is displayed as in *Figure 33*. The creator nickname, game name, mod name, mod description and size are shown.

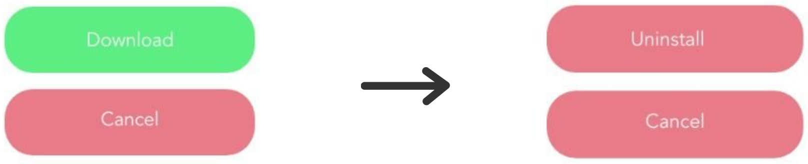


*Figure 34: Flow of Button for Different Cases*

From the information screen, users can download the mod and uninstall the mod if they have already downloaded. The change in buttons can be seen from *Figure 34*.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @m_ID refers to the id of the mod and will be taken from the frontend when the user clicks and navigates to the page of a particular mod.

➔ Display mod information

```
SELECT  u.u_name, vg.g_name, m.m_name

FROM User u, Video_Game vg, Mod m, for_m f, builds m

WHERE u.a_ID = b.a_ID AND b.m_ID = m.m_ID AND m.m_ID = f.m_ID

            AND f.g_ID = vg.g_ID;
```

➔ If download is pressed

```
INSERT INTO downloads

VALUES (@a_ID, @m_ID);
```

➔ If uninstall is pressed

```
DELETE FROM downloads

WHERE  a_ID = @a_ID AND m_ID = @m_ID;
```

## 5.13 Friends Page

**User Interface Friends Page:**

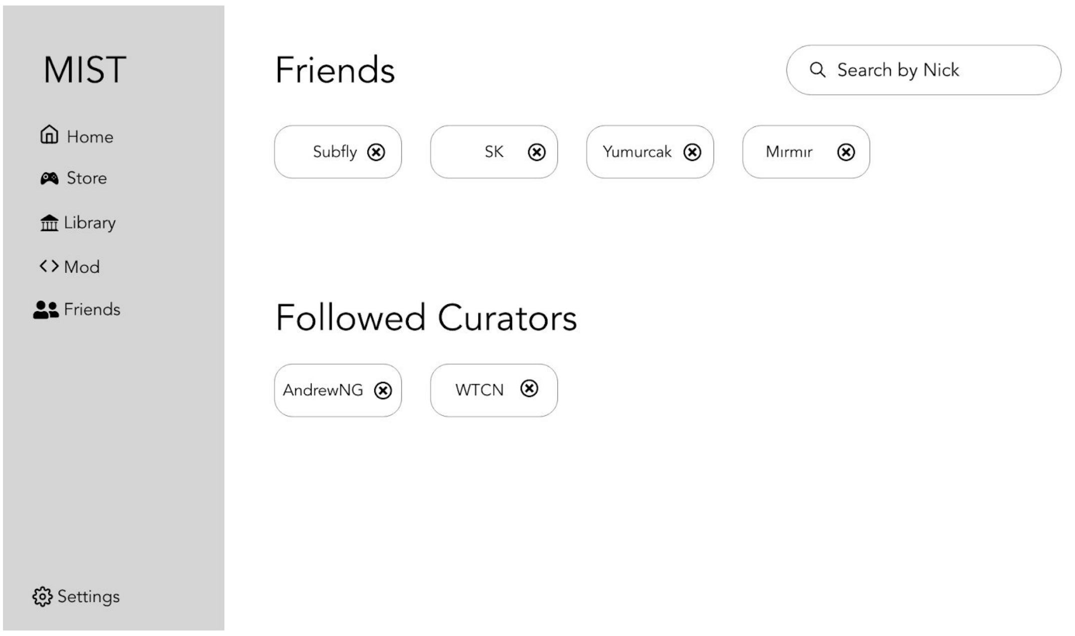


*Figure 35: Friends Page for User Types*

In the Friends page, user types can see their friends and the curators they are currently following as can be seen in *Figure 35*. Users and Curators can unfollow a friend or a curator by pressing the cancel button.

## Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @t_ID refers to the id of the friend to be unfriended and the value @c_ID refers to the id of the curator to be unfollowed. Both values are taken from the actions and selections of the user.

➔ Unfollow selected curator and unfriend with the selected user.

```
DELETE FROM friendship
WHERE(starter=@a_ID AND target=@t_ID)
        OR (target=@a_ID AND starter=@t_ID);


DELETE FROM followed_by(c_ID, a_ID)
VALUES(c_ID=@c_ID AND a_ID=@a_ID);
```

## User Interface Search Bar:

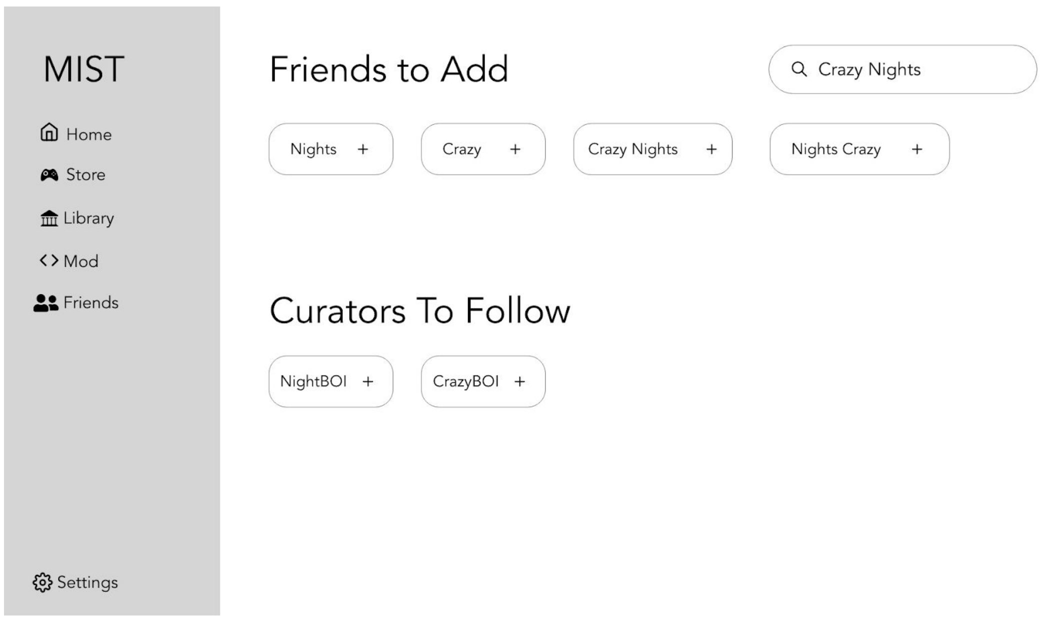


*Figure 36: Search Results Example Page*

From the search field, users can search a word to find related users and curators for adding as friends or following as can be seen from the example page in *Figure 36*.

### Corresponding SQL Statements:

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open. The value @target_ID refers to the id of the user to be friended and the value @c_ID refers to the id of the curator to be followed. Both values are taken from the actions and selections of the user. The value @key refers to the name of the friend to be added or curator to be followed and written in the search field by the user.

➔ Display the user’s friends and curators the user follows.

```
SELECT u2.nick_name
FROM friendship f, User u1, User u2
WHERE (f.target = u1.a_ID AND f.starter = u2.a_ID) AND u1.a_ID = @a_ID
          AND u2.a_ID <> @a_ID) OR
         (f.target = u2.a_ID AND f.starter = u1.a_ID) AND u1.a_ID = @a_ID
          AND u2.a_ID <> @a_ID);


SELECT u.nick_name
FROM followed_by f, User u
WHERE f.c_ID = u.a_ID, f.a_ID = @a_ID;
```

➔ Display the other users the user is not friend and curators the user does not follow. Also add friends and follow curators.

```
SELECT u3.nick_name
FROM User u3, friendship f
WHERE f.target = u3.a_ID OR f.starter = u3.a_ID
MINUS
```

```
SELECT u2.nick_name

FROM friendship f, User u1, User u2

WHERE (f.target = u1.a_ID AND f.starter = u2.a_ID) AND u1.a_ID = @a_ID

           AND u2.a_ID <> @a_ID) OR

          (f.target = u2.a_ID AND f.starter = u1.a_ID) AND u1.a_ID = @a_ID

           AND u2.a_ID <> @a_ID);


SELECT u1.nick_name

FROM Curator c, User u1

WHERE c.a_ID = u1.a_ID

MINUS

SELECT u2.nick_name

FROM followed_by f, User u2

WHERE f.c_ID = u2.a_ID, f.a_ID = @a_ID;


INSERT INTO friendship(starter, target)

VALUES(@a_ID, @target_ID);


INSERT INTO followed_by(c_ID, a_ID)

VALUES(@c_ID, @a_ID);
```

➔ For displaying the search result, the following query can be applied to the results of the select queries above.

```
SELECT *

FROM [query]

WHERE nick_name like '%@key%';
```

## 5.14 Settings Page

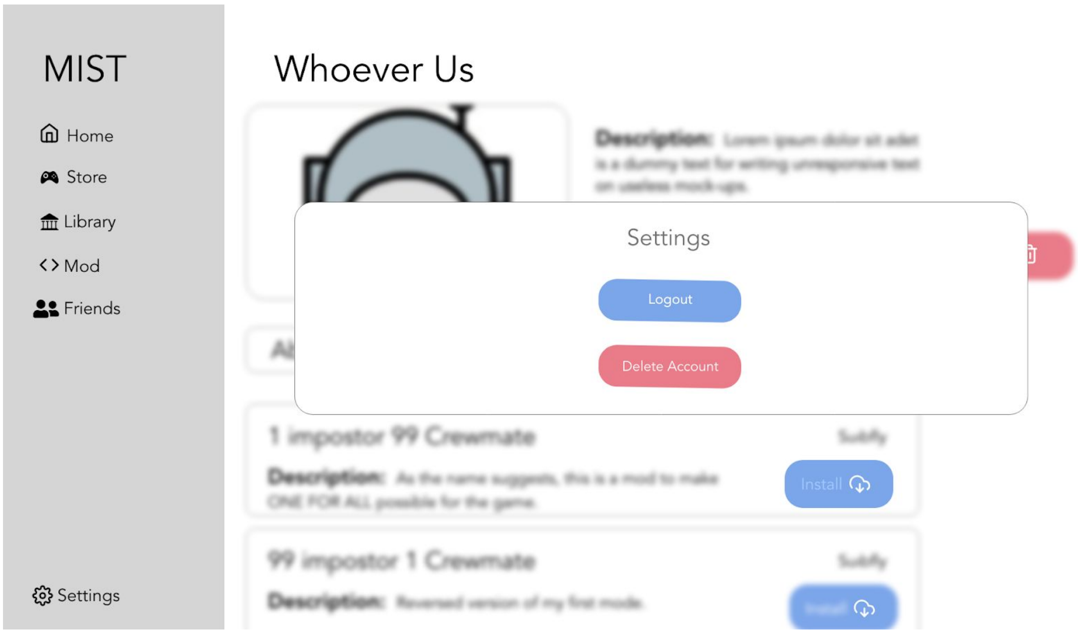


*Figure 37: Settings Screen for Logging Out*

**Corresponding SQL Statements:**

The value @a_ID refers to a_ID and points to the account id of the user and stored during the site is open.

➔ For deleting an account.

```
DELETE FROM Account

WHERE a_ID = @a_ID;
```

For the application, there is a Settings field at the left bottom of the screen. After clicking, the Logout and Delete Account buttons are shown as in *Figure 37*. All accounts can log out and delete their accounts from settings.

# 6 Advanced Database Components

## 6.1 Views

➔ Show user only “approved” and “published” games

```
CREATE VIEW Approved_Games AS

SELECT vg.g_ID, vg.g_name, vg.g_version, vg.g_description, vg.g_image, vg.g_price,
vg.genre, g_requirements

FROM Video_Game vg, publish p

WHERE vg.g_ID = p.g_ID;
```

➔ Show user only his/her friends

```
CREATE VIEW friends AS

SELECT u2.nick_name

FROM friendship f, User u1, User u2

WHERE (f.target = u1.a_ID AND f.starter = u2.a_ID) AND u1.a_ID = @a_ID

         AND u2.a_ID <> @a_ID) OR

        (f.target = u2.a_ID AND f.starter = u1.a_ID) AND u1.a_ID = @a_ID

         AND u2.a_ID <> @a_ID);
```

## 6.2 Assertions

➔ Wallet balance can not be less than 0.

```
CREATE ASSERTION balance_constraint

CHECK (NOT EXISTS

                (SELECT *

                FROM Wallet w

                WHERE w.balance < 0 ));
```

➔ Game price can not be less than 0.

```
CREATE ASSERTION price_constraint

CHECK (NOT EXISTS

                (SELECT *

                FROM Video_Game vg

                WHERE vg.g_price < 0 ));
```

## 6.3 Triggers

➔ If a user wants to buy a new game, the system will use the trigger of checking if the user has enough money in his/her Wallet.

```
CREATE TRIGGER balance_check AFTER green UPDATE ON Wallet

REFERENCING NEW ROW AS nrow

FOR EACH ROW

WHEN(nrow.balance < 0)

BEGIN

ROLLBACK

END;
```

## 6.4 Constraints

Our system will impose the following constraints:

- Comments and reviews can only be made on games that are downloaded or games that are in the bought subscription package.
- Returning a game can only be possible if the date limit to return the game has not been passed.
- Only the games that are bought can be downloaded.
- Whether the sign up email address, phone number for both user and company, nickname of the user and name of the company already exists in the database should be checked.
- A user has to be logged in to an account to use the system.
- A comment/review can be deleted by only the writer of the comment/review.
- A user can only see the games that are published in the system.
- Users can only leave one comment for each game they own.
- Developers cannot update games of other developers.
- Each individual has to select their login type in order to be successfully authorized.

## 6.5 Stored Procedures

- Getting the average rate of an app.

```
CREATE PROCEDURE get_avg_rating_of_app(IN g_id INT, OUT avg_rating NUMERIC(2,1))

BEGIN

SELECT AVG(rate) AS avg_rating

FROM rates r

WHERE r.g_ID= g_id;

END
```

# 7 Website

The website of the project along with report and the repository that holds the source code can be found following:

https://subfly.github.io/Stream-Site/#/

# 8 References

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database system concepts, 6th ed. New York: McGraw-Hill, 2010. [Online]. Available: http://www.db-book.com/