BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 353 -Database Systems

Fall 2016

Wapour.Ware (Gaming Platform)

Design Document

Group 9

Tuğrulcan Elmas	21301913
Cihan Eryonucu	21301158
Aras Heper	21302248
Fırat Özbay	21201683

Website: arasheper.github.io/wapour.ware

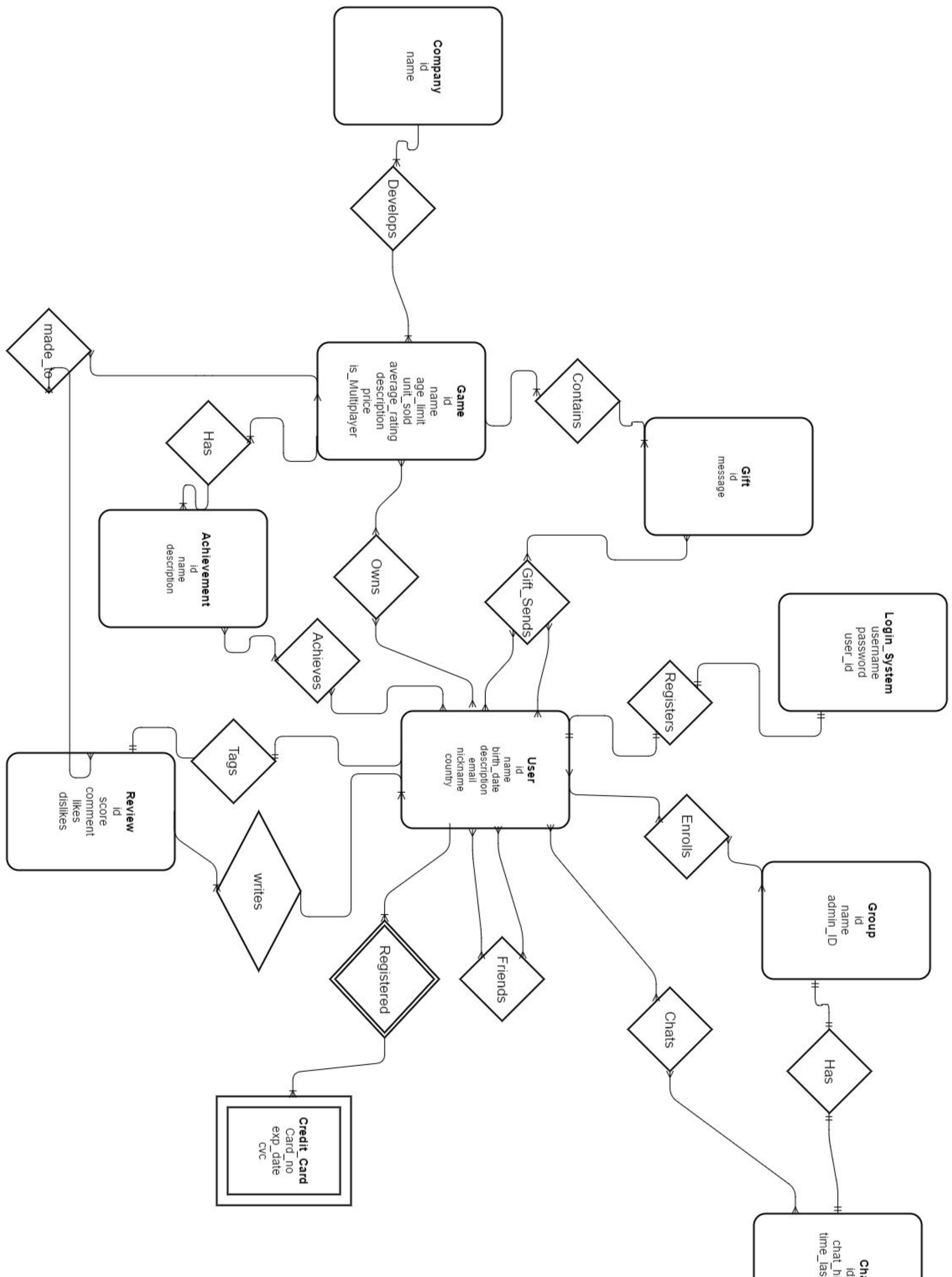
Table Of Contents

1. Introduction

wapour.ware (W.W) is a system that integrates an online gaming platform, a market for selling game products and a social network for gamers. In this platform the players are able to buy games, play games, rate them and write reviews about them. W.W categorizes the games by genre, multiplayer availability. It also stores the games' general information such as price of the game, brief description and age limit as well as statistical data such as average rating. The system is able to sort the games by according to game data it stores. W.W also keeps track of user specific data such as their ratings, reviews and progress and achievements in the games. Additionally, it saves user data, such as their friends and the gift they send to their friends and the friends who owns the game they examine at the time. Groups of users and the members of the chatrooms they have formed are also recorded. The system will feature a web-based interface for users and administrators.

2. E/R Diagram

Notes on the diagram: If there is a triangle in the relationship this means many. If there is no triangle, just a line, that means one. So it is reverse of the slides. We specified one relations with number too so cope with misunderstandings. This is because of the visual paradigm. Another thing to mention is big red U's in columns. This U specify the attribute with unique property. Therefore, if there is a U near attribute that means that attribute is unique. Key symbol near the attribute means that attribute is primary key.
The diagram can be found on the next page.


Company
id
name
Develops
Game
id
name
age_limit
unit_sold
average_rating
description
price
is_Multiplayer
made_to
Has
Contains
Gift
id
message
Achievement
id
name
description
Owns
Gift_Sends
Login_System
username
password
user_id
Achieves
Registers
Tags
User
id
name
birth_date
description
email
nickname
country
Review
id
score
comment
likes
dislikes
Enrolls
writes
Group
id
name
admin_ID
Registered
Friends
Chats
Has
Credit_Card
Card_no
exp_date
cvc
Chat
id
chat_history
time_last_seen

3. Functionalities

A user should be able to:

- Register an account with their personal information.
- Login using his/her email address and password.
- View the list of games they own.
- View the list of total games in the store.
- View the list of their friends.
- View the list of reviews of a game.
- Review a game they own and give a score to the game.
- Like or dislike the review of another user.
- Create and join chats.
- Form or join a “group”.
- Join the chat of a specific “group”
- Save credit card information.
- Buy a game from the store with a credit card.
- Gift a game to another account.

4. Requirements

A database is required to store all these data in an efficient and effective manner. Complex user-game and user-user relationships also promotes using of databases. Database is also enhances filtering and sorting the games. Looking for which friends own a specific game requires applying query, which a database would assist in.

A back-end framework is also required to implement a server to take requests from clients and apply queries to produce the necessary response in a timely manner.

5. Limitations

Friendship is single-sided, person who became of a friend of somebody does not have to add him back. A person who does not own a game cannot play, review or rate the game that he does not own but he can rate that game’s reviews. A user who does not have a credit card cannot buy games. A user who is below 18 cannot play games which is rated 18+. A user cannot play the games which his/her country prohibited. A user cannot gift a game to a user who has the game. Chat history space will be limited. A game need to be multiplayer to be played with other people. All operations need users to be logged in. Groups only can have one administrator. To view the chat of the selected group, one needs to be in that group.

6. Relational Schemas

6.1 Company

Relation

company (<u>id</u>, name)

Functional Dependencies

ID → name

Candidate Keys

{ID}

Table Definition

Create table company (id int PRIMARY KEY AUTO_INCREMENT, name varchar(255))

6.2 Game

Game (<u>id</u>, name, genre, age_limit, unit_sold, average_rating, description, price, isMultiplayer)

Functional Dependencies

ID → name, genre, age_limit, unit_sold, average_rating, description, price, isMultiplayer)

Candidate Keys

{ID}

Table Definition

Create table game(id int PRIMARY KEY AUTO_INCREMENT, name varchar(255)

Genre name varchar(255), age int, unit sold int, average_rating float(2), description varchar(1000), price int, isMultiplayer smallint, FOREIGN KEY company_id references Company(ID))

6.3 User

User (<u>id</u>, name, birth_date, description, email, nickname, country)

Functional Dependencies

ID → name, birth_date, description, email, nickname, country

Candidate Keys

{ID}

Table Definition

Create table user(id int PRIMARY KEY AUTO_INCREMENT, name varchar(255), birth_date date, description varchar(1000), email varchar(255), nickname varchar(25), country(255))

6.4 Group

Group (<u>id</u>, name, admin_id)

Functional Dependencies

ID → name, admin_id

Candidate Keys

{ID}

Table Definition

Create table group(id int PRIMARY KEY AUTO_INCREMENT, name varchar(255), FOREIGN KEY admin_id references User(id))

6.5 Review

Relation

Review(ID, score, comment, likes, dislikes)

Functional Dependencies

ID → score, comment, likes, dislikes

Candidate Keys

{ID}

Table Definition

Create table review(id int PRIMARY KEY AUTO_INCREMENT, score int, dislikes int, likes int, FOREIGN KEY user_id references User(id), FOREIGN KEY game_id references Game(ID))

6.6 Achievement

Relation

Achievement(ID, name, description)

Functional Dependencies

ID → name, description

Candidate Keys

{ID}

Table Definition

Create table group(id int PRIMARY KEY AUTO_INCREMENT, name varchar(255), description varchar(1000), FOREIGN KEY game_id references Game(ID))

6.7 Chat

Relation

Chat(ID, chat_history, time_last_seen)

Functional Dependencies

ID → chat_history, time_last_seen

Candidate Keys

{ID}

Table Definition

Create table group(id int PRIMARY KEY AUTO_INCREMENT, chat_history varchar(1000), time_last_seen timestamp)

6.8 Gift

Relation

Gift(ID, message)

Functional Dependencies

ID → message

Candidate Keys

{ID}

Table Definition

Create table group(id int PRIMARY KEY AUTO_INCREMENT, message varchar(1000))

6.9 Achieves

Achieves(<u>GameID</u>, <u>AchievementID, UserID</u>)

Functional Dependencies

No nontrivial dependencies

Candidate Keys

{(GameID, AchievementID, UserID)}

Table Definition

Create table achieves(FOREIGN KEY gameId references Game(ID), FOREIGN KEY achievementId references Achievement(ID), FOREIGN KEY userId references User(ID))

6.10 Writes

Writes(<u>GameID</u>, <u>UserID, ReviewID</u>)

Functional Dependencies

No nontrivial dependencies

Candidate Keys

{(GameID, UserID, ReviewID)}

Table Definition

Create table achieves(FOREIGN KEY gameId references Game(ID), FOREIGN KEY userId references User(ID), FOREIGN KEY reviewId references Review(ID))

6.11 Tags

Tags(<u>GameID</u>, <u>UserID,</u> R_Type)

Functional Dependencies

GameID, UserID → R_Type

Candidate Keys

{(GameID, UserID)}

Table Definition

Create table tags(FOREIGN KEY gameId references Game(ID), FOREIGN KEY userId references User(ID), R_Type int)

6.12 Login_System

Login_System(<u>username</u>, password, user_id)

Functional Dependencies

username → password, user_id

User_id → password, username

Candidate Keys

{username}

Table Definition

Create table login_system(username varchar(255) PRIMARY KEY , password varchar(255), FOREIGN KEY user_id references User(ID))

6.13 Friends

Friends(<u>UserID</u>, <u>UserID2</u>)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(UserID, UserID2)}

Table Definition

Create table tags(FOREIGN KEY userId references User(ID), FOREIGN KEY userId2 references User(ID))

6.14 Chats

Chats(ChatID, UserID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(ChatID, UserID)}

Table Definition

Create table tags(FOREIGN KEY userId references User(ID), FOREIGN KEY userId2 references User(ID))

6.15 Owns

Owns(GameID, UserID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(GameID, UserID)}

Table Definition

Create table owns(FOREIGN KEY gameId references Game(ID), FOREIGN KEY userId references User(ID))

6.16 Gift_Sends

Gift_Sends(GiftID, SenderID, ReceiverID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(GiftID, SenderID, ReceiverID)}

Table Definition

Create table gift_sends(FOREIGN KEY giftId references Gift(ID), FOREIGN KEY userId references User(ID), FOREIGN KEY userId2 references User(ID))

6.17 Contains

Contains(GiftID, GameID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(GiftID, GameID)}

Table Definition

Create table contains(FOREIGN KEY giftId references Gift(ID), FOREIGN KEY gameId references Game(ID))

6.18 Group_Chat

Group_Chat(GroupID, ChatID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(GroupID, ChatID)}

Table Definition

Create table group_chat(FOREIGN KEY groupId references Group(ID), FOREIGN KEY userId references User(ID))

6.19 Enrolls

Enrolls(UserID, GroupID)

Functional Dependencies

No Nontrivial Functional Dependency

Candidate Keys

{(UserID, GroupID)}

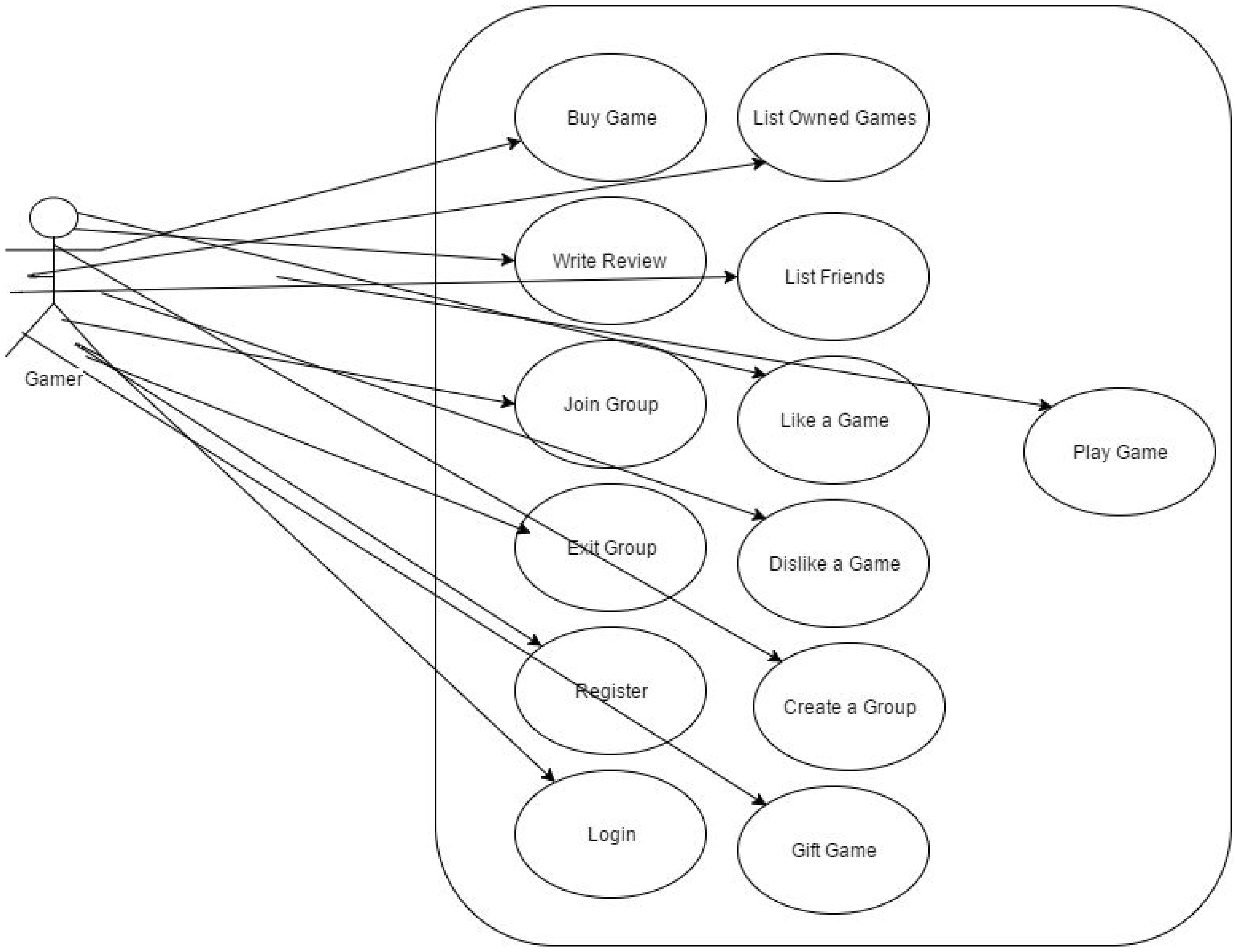
Table Definition

Create table enrolls(FOREIGN KEY userid references User(ID), FOREIGN KEY groupId references Group(ID))

7. Functional Components

The modules of each functional component should be determined identifying their input and output parameters, data structures, and high-level algorithms. Use-cases/scenarios for potential user groups are provided.

7.1 Use Case Diagrams/Scenarios



- User can buy a game
- User can write reviews to a game
- User can join group
- User can exit group

- User can register to the WapourWare to open and account
- User can login to the system
- User can list his/her owned games
- User can list his friends
- User can like a game
- User can dislike a game
- User can create a gaming group, it will automatically generate a group chat for that group
- User can gift a game to another user
- User can launch a game from the system

7.2 Algorithms

7.2.1 Buy Game

The user can buy a game via credit card. Everytime the user buys a game, the unit_sold attribute of the game is increased by one, then the user is given access to rate and write a review about the game. The game will be available in the games the user owns. The user will be able to play that game and unlocks (achieves) its achievements. A tuple will be added to owns id with corresponding user id and game id.

7.2.2 Writing Review

The user can write a review to game from the game's page. In the review, he/she can write comments for a limited characters and report his score. A new review will be added to review table with the corresponding game and user id.

7.2.3 Tagging Review

The other users can tag the existing reviews as "like" or "dislike". When they do, the review's like/dislike number will be incremented accordingly. A corresponding tuple will also be added to Tags table.

7.2.4 Groups

The user will be able to form groups. The user who forms the group will be admin, he will have privileges access such as disapproving join requests and kick players. A new user is added to group if that user requests a join and the admin approves it. User can exit the group whatever he wants and when he does, the corresponding tuple involving the user will be deleted.

7.2.5 Registration

The user can registration by filling the registration form. A new tuple will be added to User table by the form's elements filled in. Also, new tuple will added to login_system with username and password of the user which will make users to login to the systems

7.2.6 Login

The user can login to system by writing its username and password.. The system validates the data by using the a view which validates those data by checking the attributes of the user with that username and userID.

7.2.7 List Games

The user is able see the games he/she owns. The system retrieves the game list by querying the owns table and retrieve the game info of the corresponding game id.

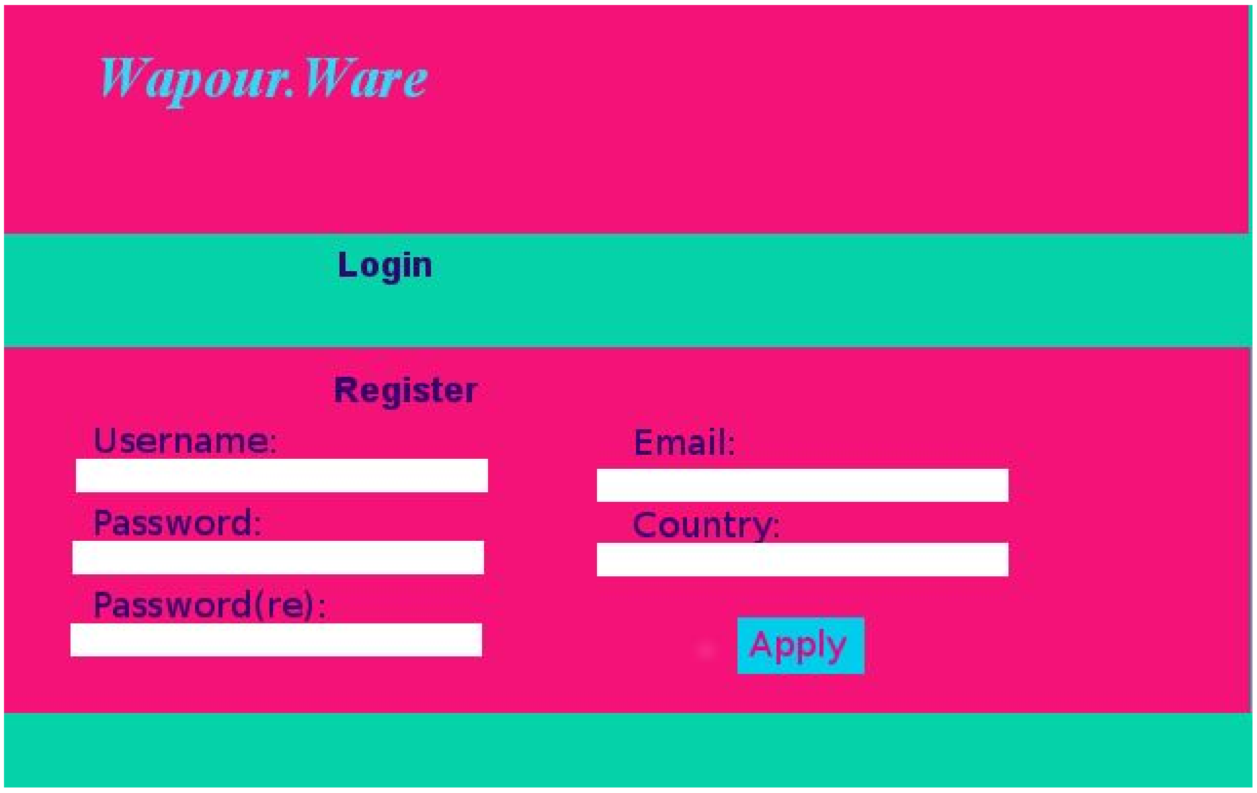
7.2.8 List Friends

The user is able to see the friends he/she has. The system retrieves the friends list by querying the friends table and retrieve the friend info from the user table by the corresponding user id.

7.3 Data Structures

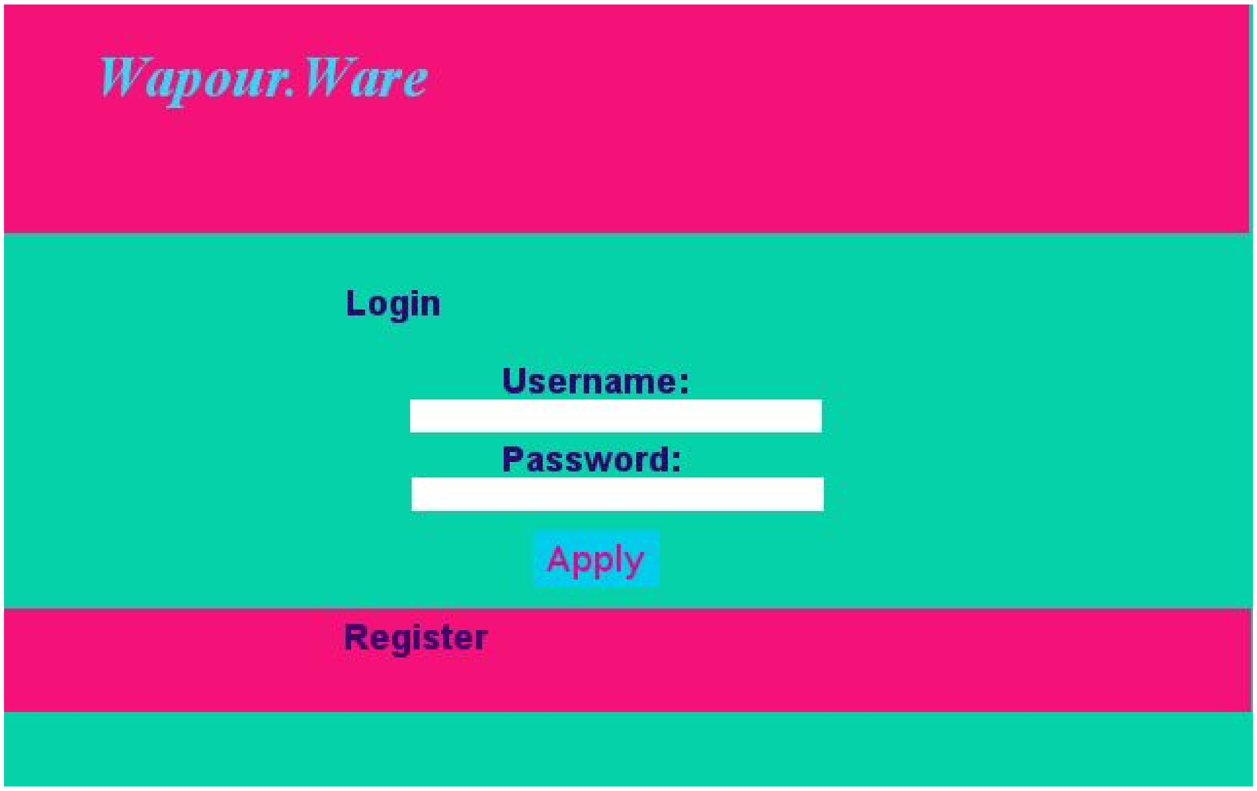
All necessary information will be stored in database. We will not use any text file to store data because our program does not need big data. The only thing that requires data is chat history and we will limit the history so that only recent chat will be stored. Date and time data types will be obtained from MySQL

8. UI and SQL Queries Rel.



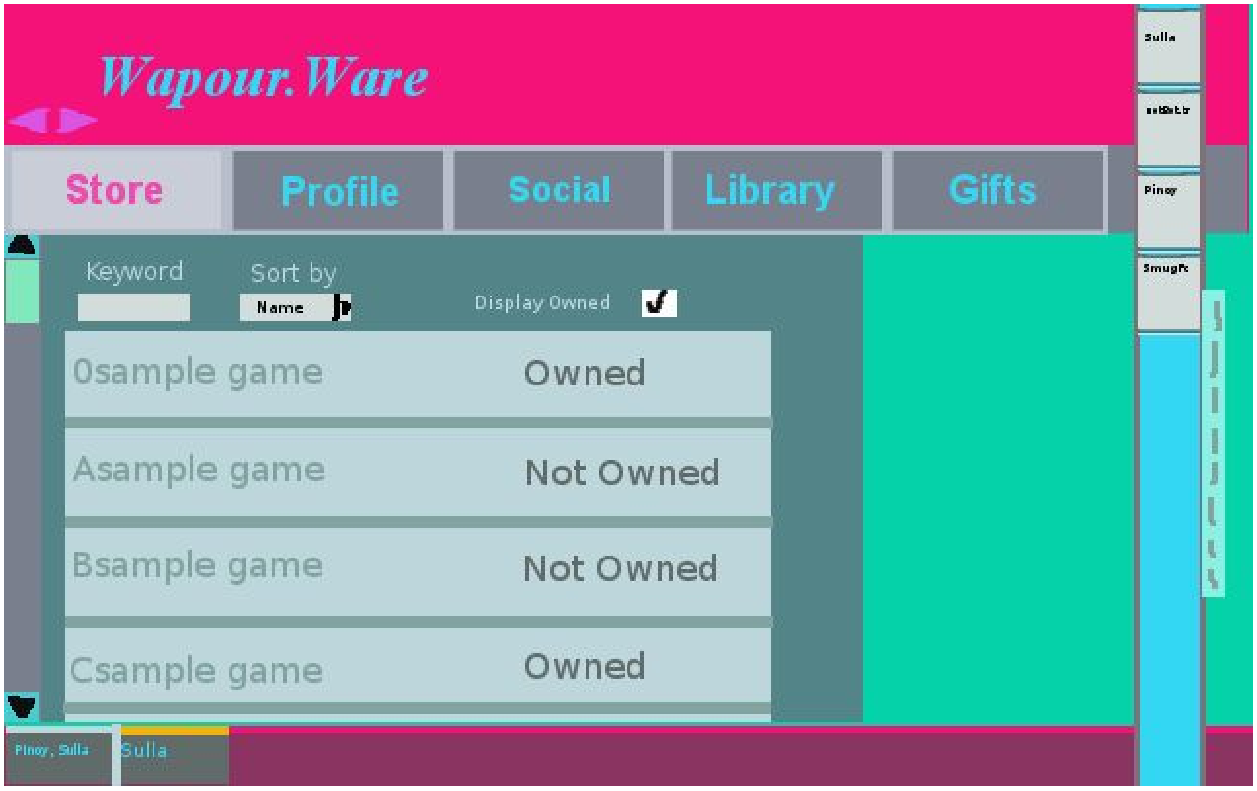
Inputs @email, @password, @password2, @name, @country
Process: The user enters their profile details in order to register to the system.
SQL Statements:

```
UPDATE user
SET email = @email, password = @password, name = @name, country = @country
WHERE @password = @password2 AND NOT EXISTS
        (SELECT FROM user WHERE email = @email OR name = @name)
```



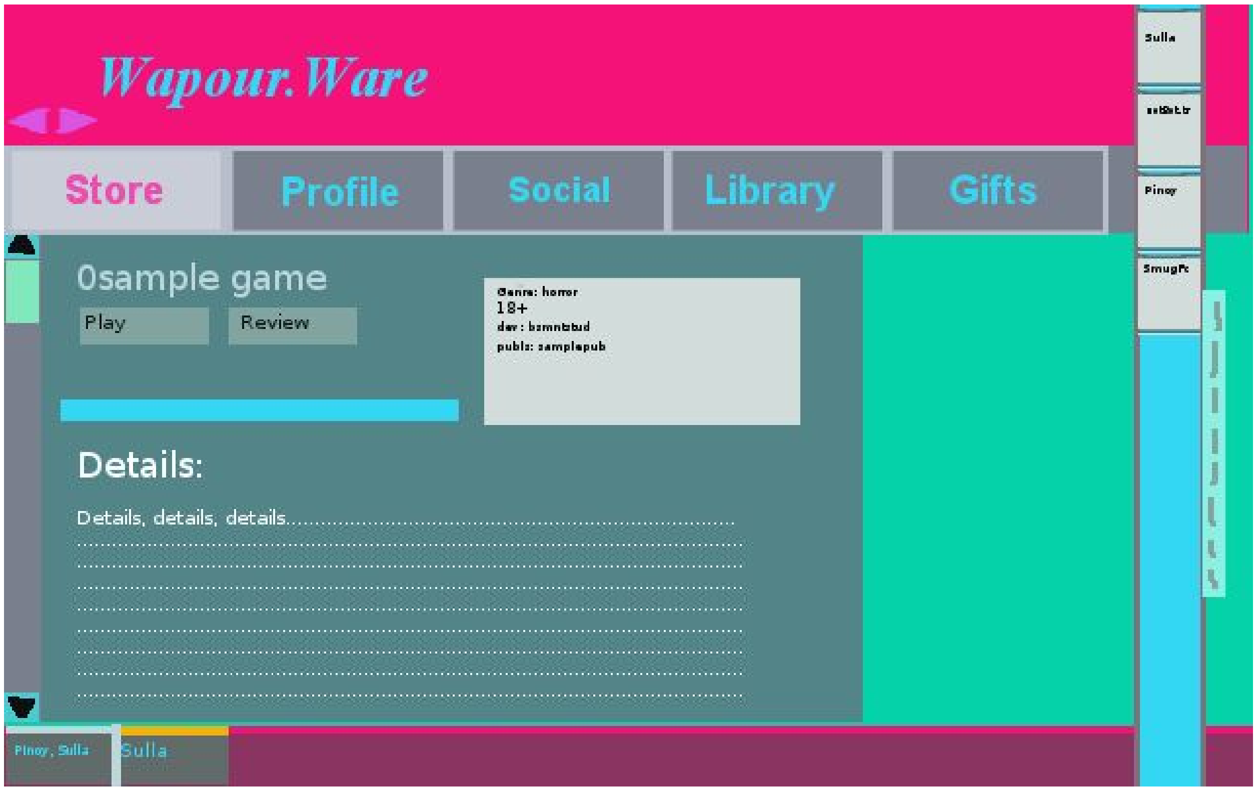
Inputs @email, @password
Process: The user enters their email and password to login to the system.
SQL Statements:

```
SELECT email, password
FROM user
WHERE email= @email AND password = @password
```



Inputs @keyword, @sortby, @displayOwned, @userID
Process: The user searches for games on the marketplace and games are listed on the search menu.
SQL Statements:

```
SELECT *
FROM (SELECT Game.name AS gameName, ‘Owned’
          FROM Game JOIN Owns ON Owns.gameID = Game.gameID
         WHERE userID = @userID AND @displayOwned = true ) AS t1
         JOIN (( SELECT Game.name AS gameName, ‘Not Owned’
                 FROM Game JOIN Owns ON Owns.gameID = Game.gameID
                 WHERE userID = @userID ) AS t2  RIGHT JOIN
                 Game ON Game.gameID = t2.gameID WHERE t2.gameID IS NULL ) AS t3 ON
                 t1.gameName = t3.gameName
ORDER BY gameName ASC
WHERE gameName LIKE ‘%@keyword%’
```

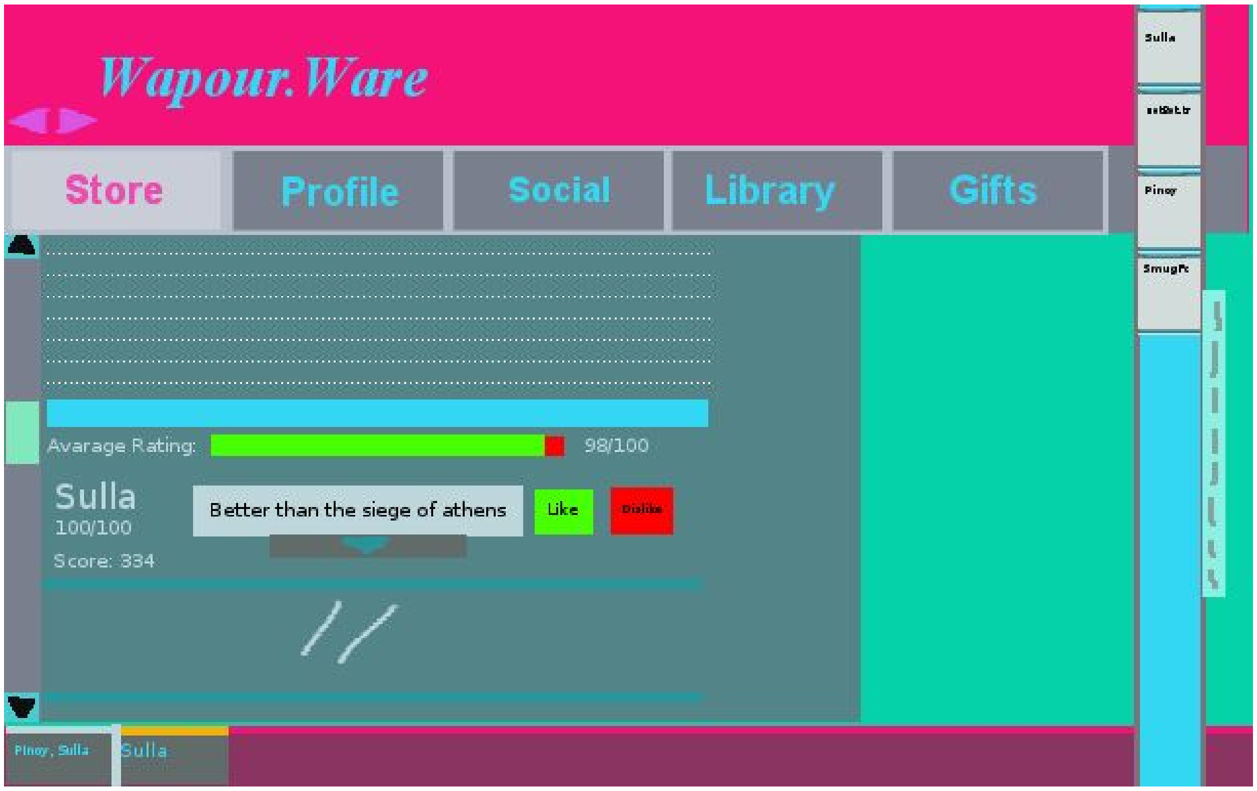


Inputs @GameID, @userID
Process: The user is looking at a game on the store
SQL Statements:
Selecting the game on the store

```
SELECT *
FROM Game
WHERE Game.gameID = @gameID
```

The play button and review button is displayed if the user owns the game otherwise buy button is displayed

```
SELECT userDoesntOwnIt
FROM (SELECT * FROM Game JOIN Owns ON Owns.gameID = Game.gameID
        WHERE userID = @userID ) AS t2  RIGHT JOIN Game
        ON Game.gameID = t2.gameID
        WHERE (t2.gameID IS NULL) AS userDoesntOwnIt
```



Inputs @GameID, @userID

Process: The user is looking at a game on the store

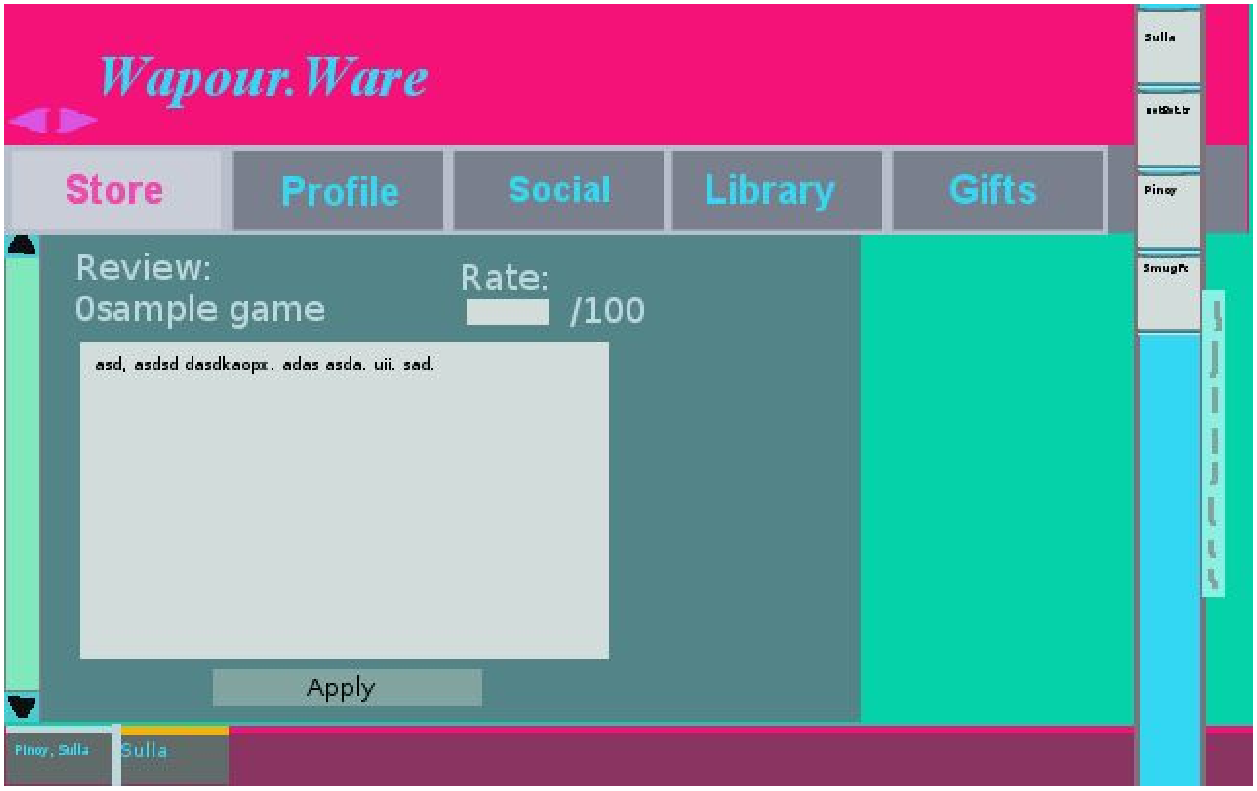
SQL Statements:

Selecting the game on the store

```
SELECT *
FROM Game
WHERE Game.gameID = @gameID
```

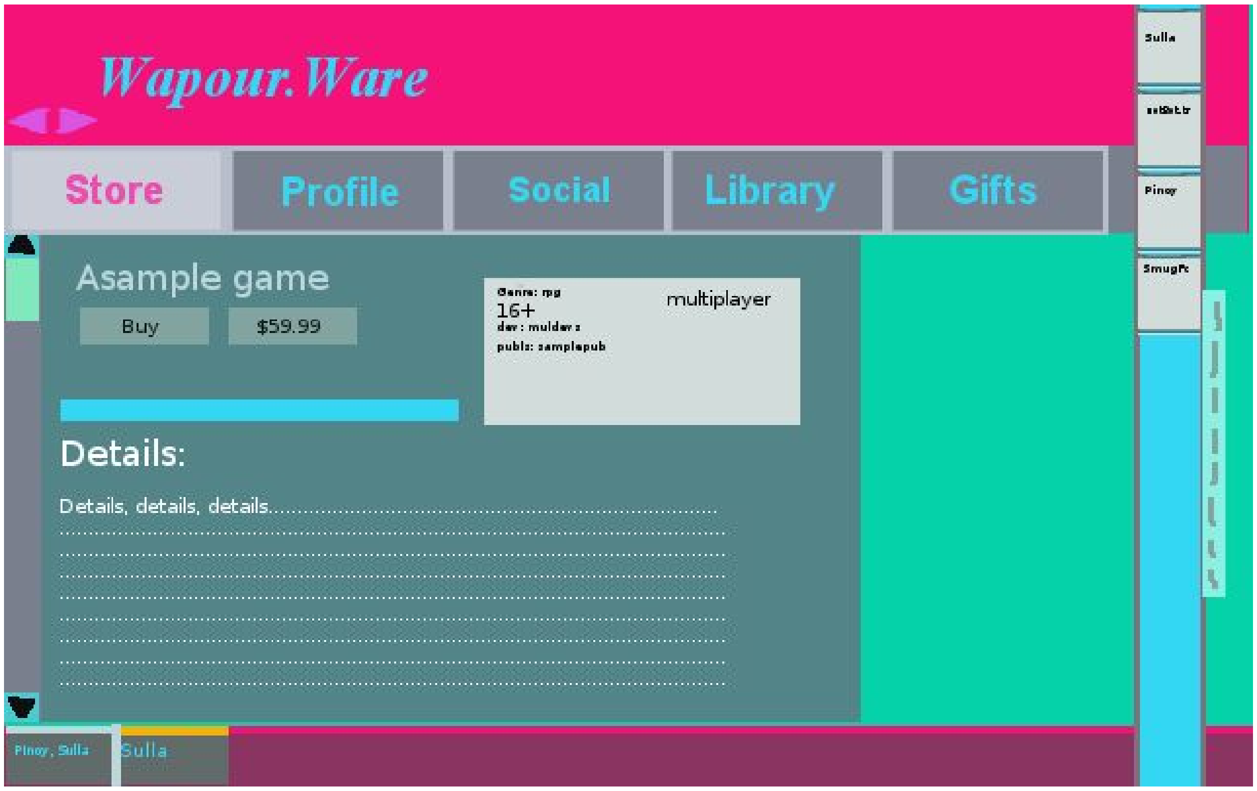
The play button and review button is displayed if the user owns the game

```
SELECT userDoesntOwnIt
FROM (SELECT * FROM Game JOIN Owns ON Owns.gameID = Game.gameID
        WHERE userID = @userID ) AS t2  RIGHT JOIN Game
        ON Game.gameID = t2.gameID
        WHERE (t2.gameID IS NULL) AS userDoesntOwnIt
```



asd, asdsd dasdkaopx. adas asda. uii. sad.

Inputs @userID, @Score, @reviewID, @gameID, @Comment
Process: The user reviews the game and gives a score.
SQL Statements:
UPDATE Review
SET ReviewID = @reviewID, Score = @Score, Comment = @Comment
UPDATE Writes
SET ReviewID = @reviewID, userID = @userID
UPDATE Made_To
UPDATE ReviewID = @reviewID, gameID = @gameID

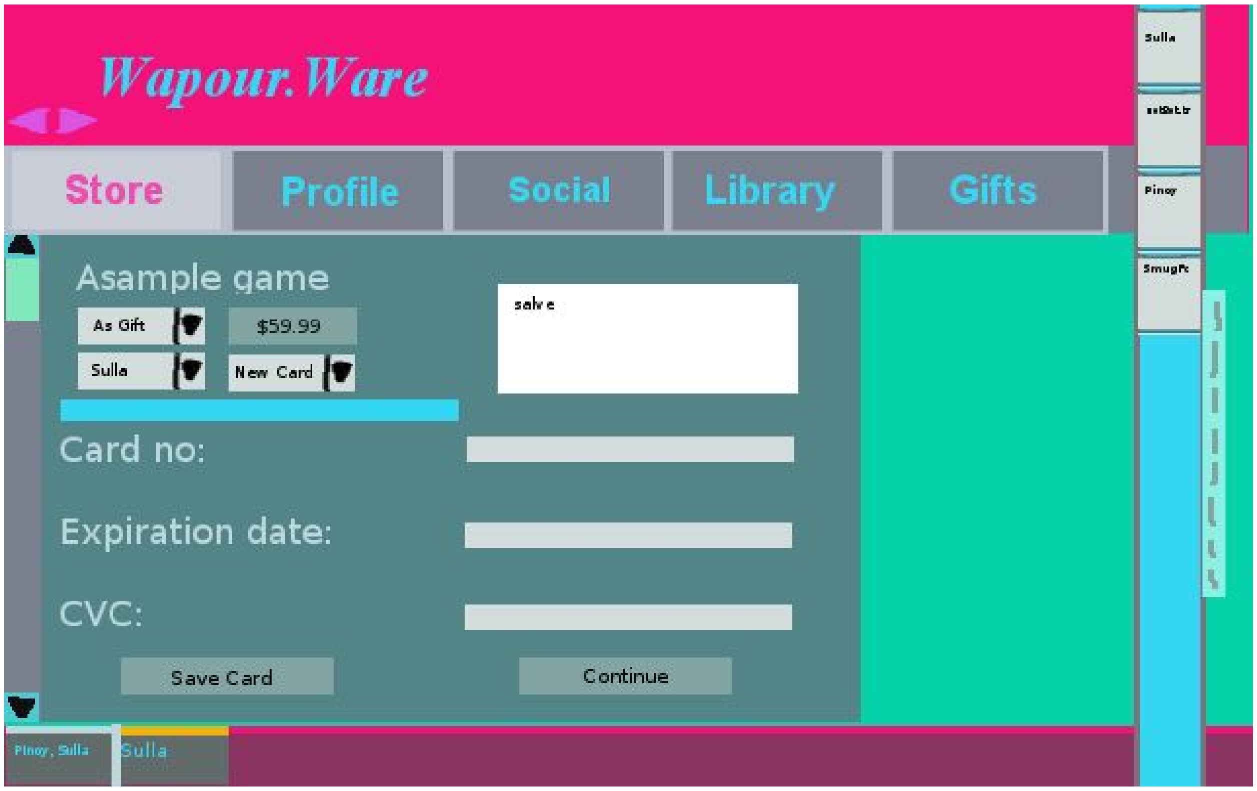


Inputs @GameID, @userID
Process: The user is looking at a game on the store
SQL Statements:
Selecting the game on the store

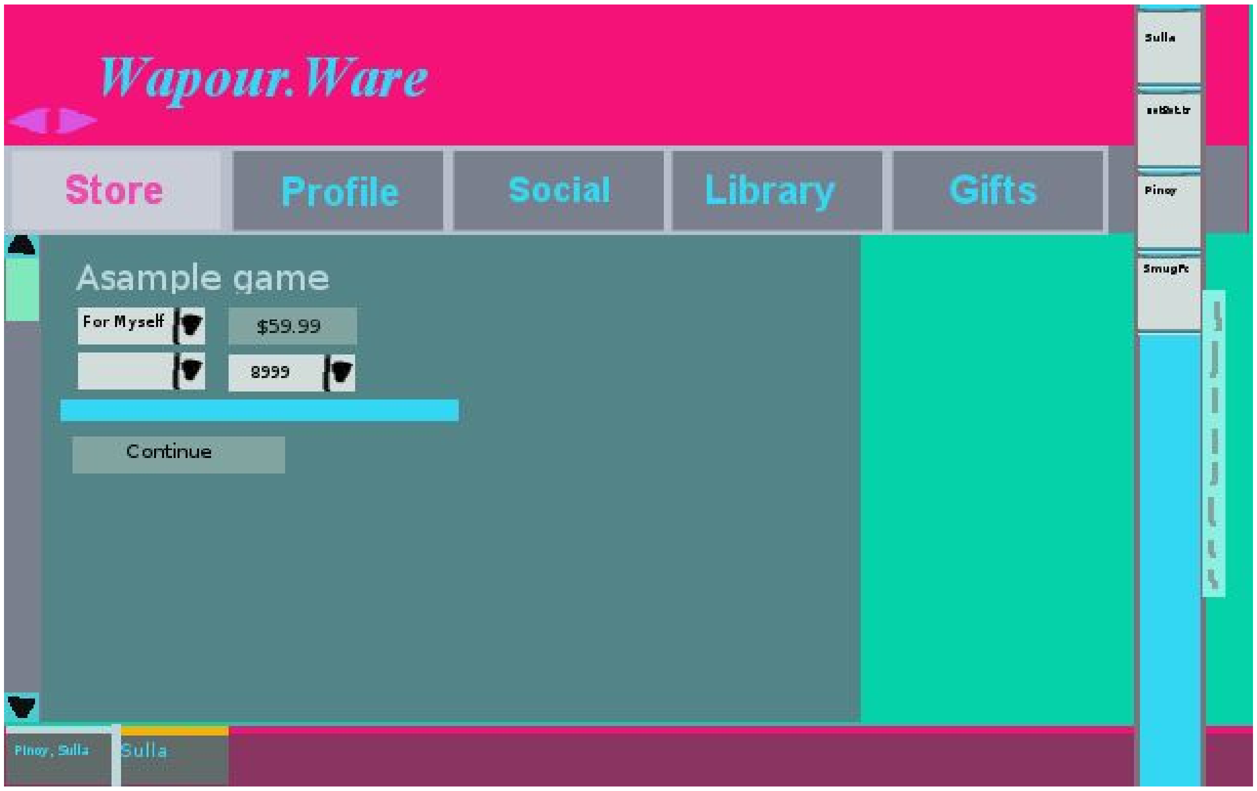
```
SELECT *
FROM Game
WHERE Game.gameID = @gameID
```

The buy button and the price is displayed if the user doesn't own the game

```
SELECT userDoesntOwnIt, t2.price
FROM (SELECT * FROM Game JOIN Owns ON Owns.gameID = Game.gameID
        WHERE userID = @userID ) AS t2  RIGHT JOIN Game
        ON Game.gameID = t2.gameID
        WHERE (t2.gameID IS NULL) AS userDoesntOwnIt
```



Inputs @userID, @giftedUserID, @CreditCardNo, @ExpirationDate, @CVC, @GiftID, @gameID, @message
Process: The user is buying a game on the store as a gift and Uses an unregistered credit card
SQL Statements:
Save Card is clicked on the buy menu
UPDATE CreditCard
SET CardNo = @CreditCardNo, Exp_Date = @ExpirationDate, CVC = @CVC,
UPDATE Registered
SET Credit_Card_No = @CreditCardNo, userID = @userID
After saving the card,must send the buy request data to the bank
SELECT Card_no, exp_date,cvc, t1.price
FROM Credit_Card JOIN Registered ON Credit_Card_No = Card_No
LEFT JOIN(SELECT price FROM Game WHERE gameID = @GameID) AS t1
WHERE UserID = @UserID
As Gift is Selected
UPDATE Gift_Sends
SET GiftID = @GiftID, SenderID = @userID, RecieverID = @giftedUserID
UPDATE Gift
SET GiftID = @GiftID, message = @message
UPDATE Contains
SET GiftID = @GiftID, gameID = @gameID

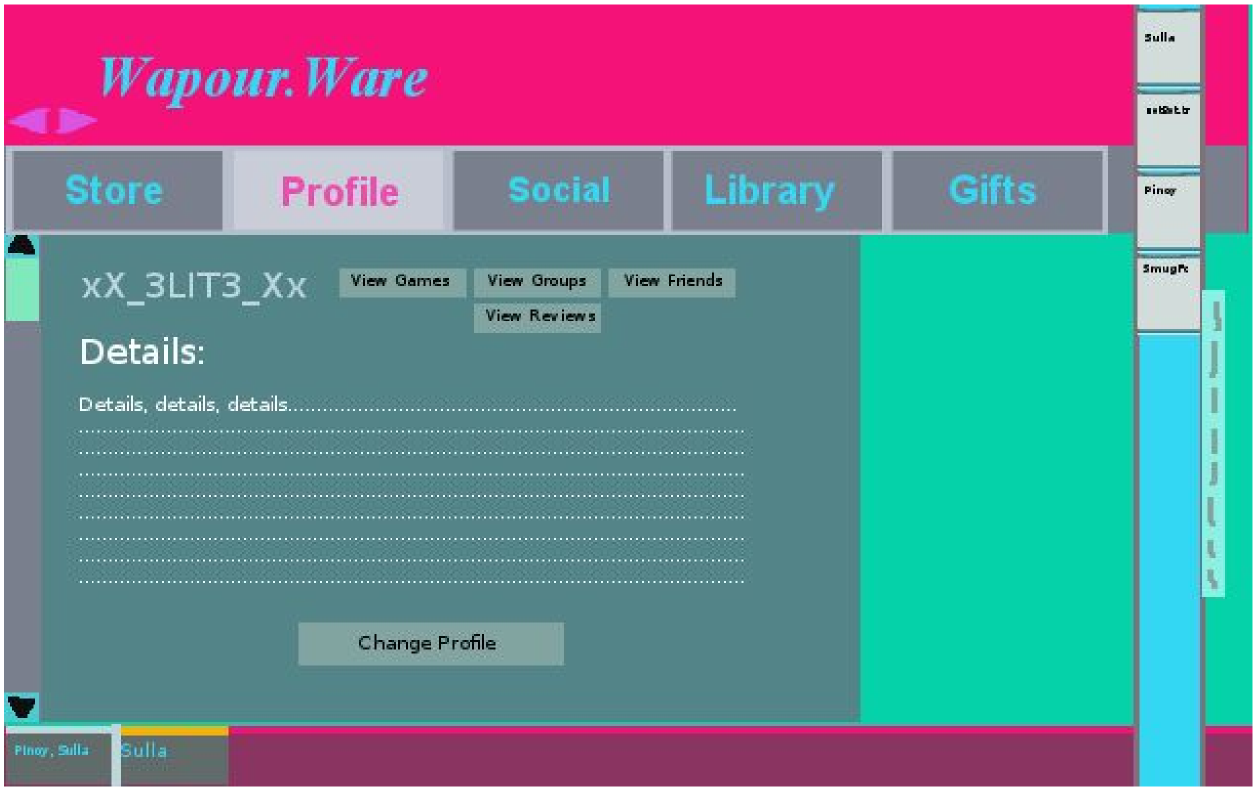


Inputs @userID, @CreditCardNo, @ExpirationDate, @CVC, @GiftID, @gameID,
Process: The user is buying a game on the store as a gift
SQL Statements:
An existing Credit Card is Selected and the game price is selected in order to send buy request to bank

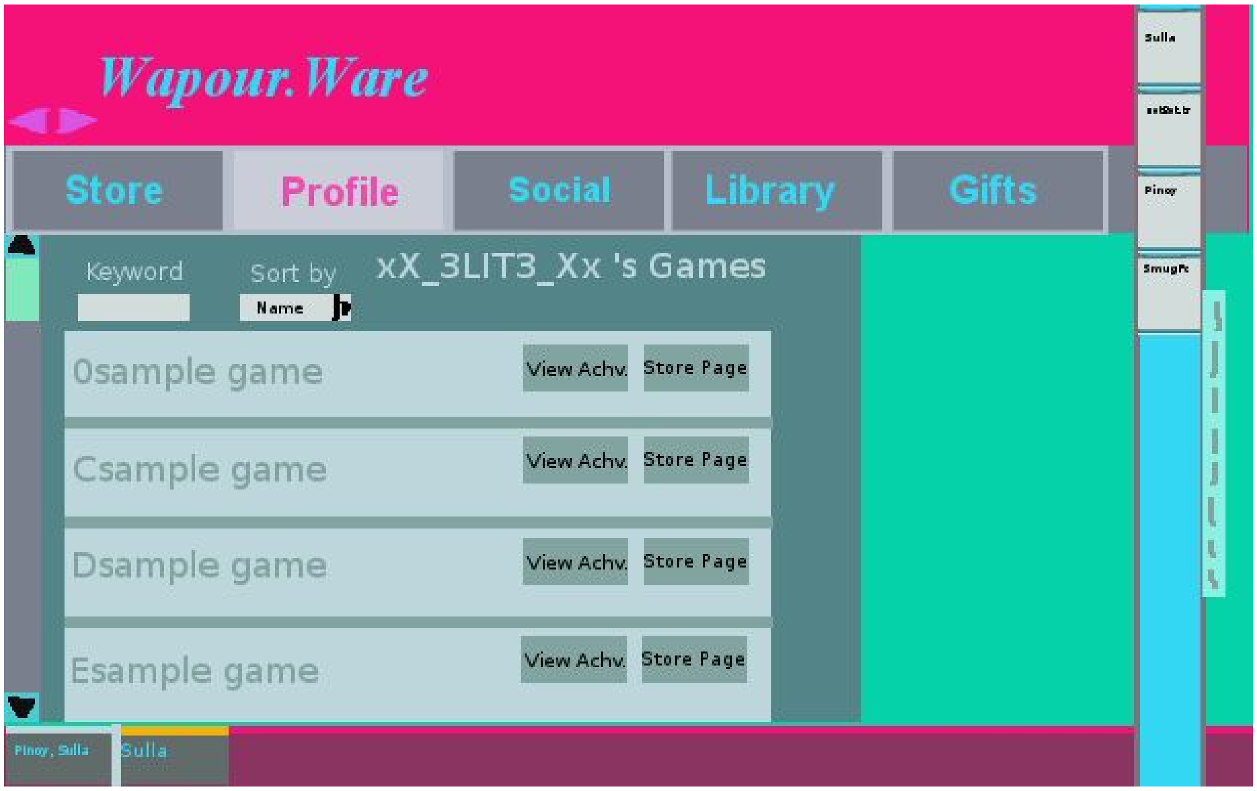
```
SELECT Card_no, exp_date,cvc, t1.price
FROM Credit_Card JOIN Registered ON Credit_Card_No = Card_No
LEFT JOIN(SELECT price FROM Game WHERE gameID = @GameID) AS t1
WHERE UserID = @UserID
```

As Gift is not Selected

```
UPDATE Owns
SET gameID = @gameID, userID = @userID
```

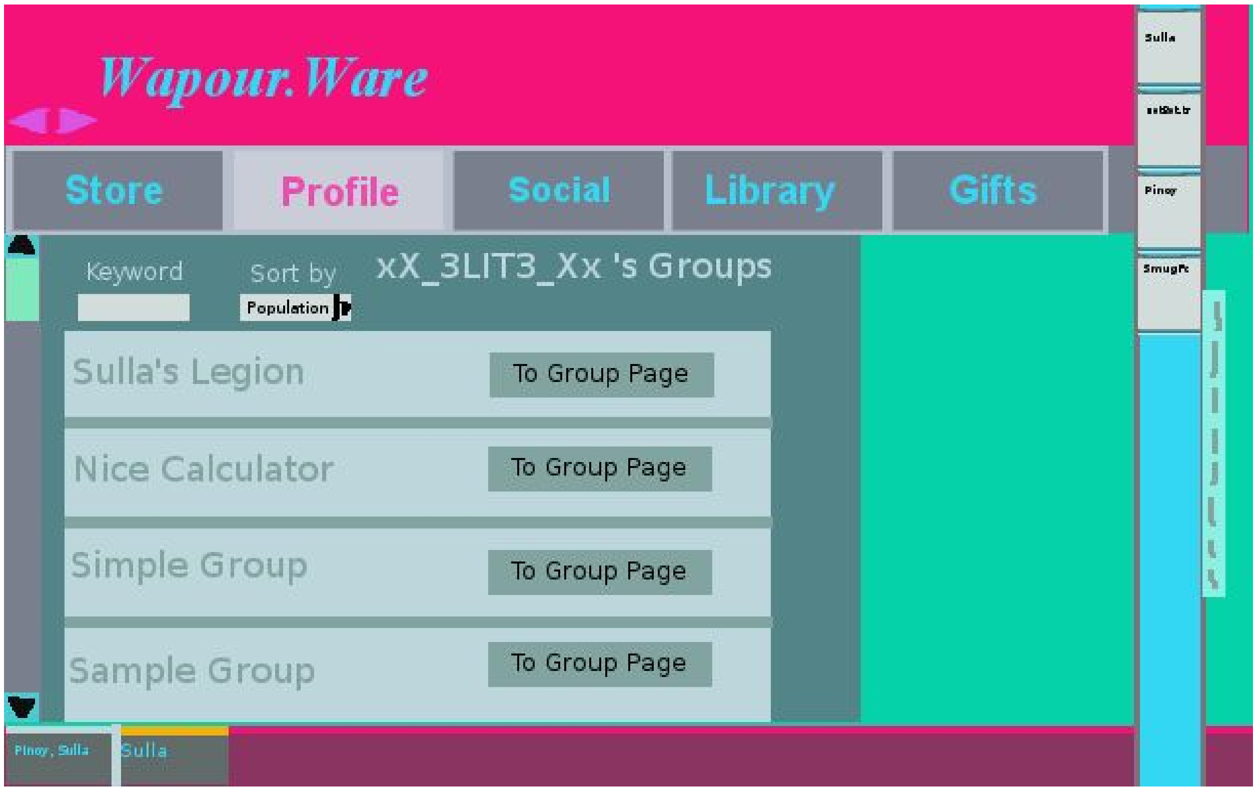


Inputs @userID
Process: The user is looking at their profile on the application
SQL Statements:
Selecting the user and showing the user data
SELECT *
FROM Users
WHERE User.userID = @userID

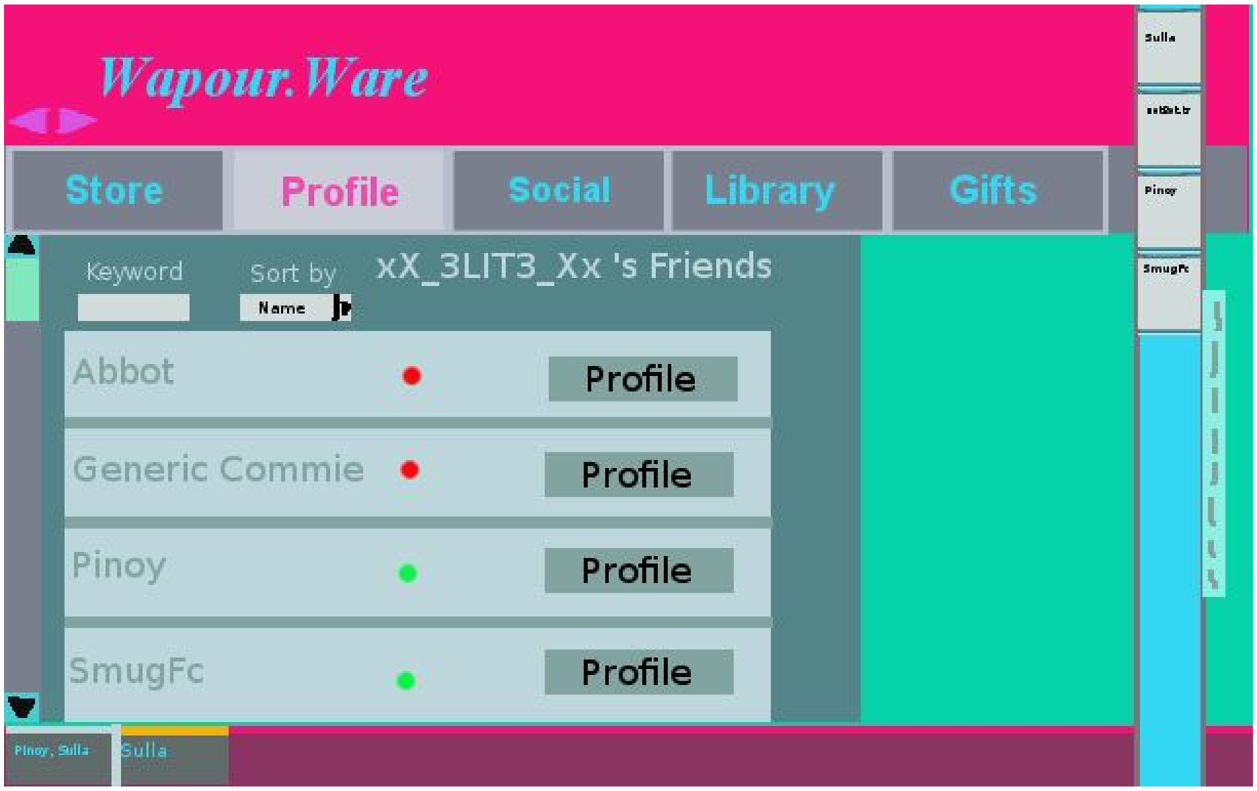


Inputs @userID,@keyword
Process: The user is looking at their games list on the application
SQL Statements:

```
SELECT Game.name, Has.AchievementID, Game.gameID
FROM Owns JOIN Game ON Owns.gameID = Game.gameID
JOIN Has ON Game.gameID = Has.gameID
WHERE Owns.userID = @userID
ORDER BY Game.name ASC
WHERE Game.name LIKE '%@keyword%'
```

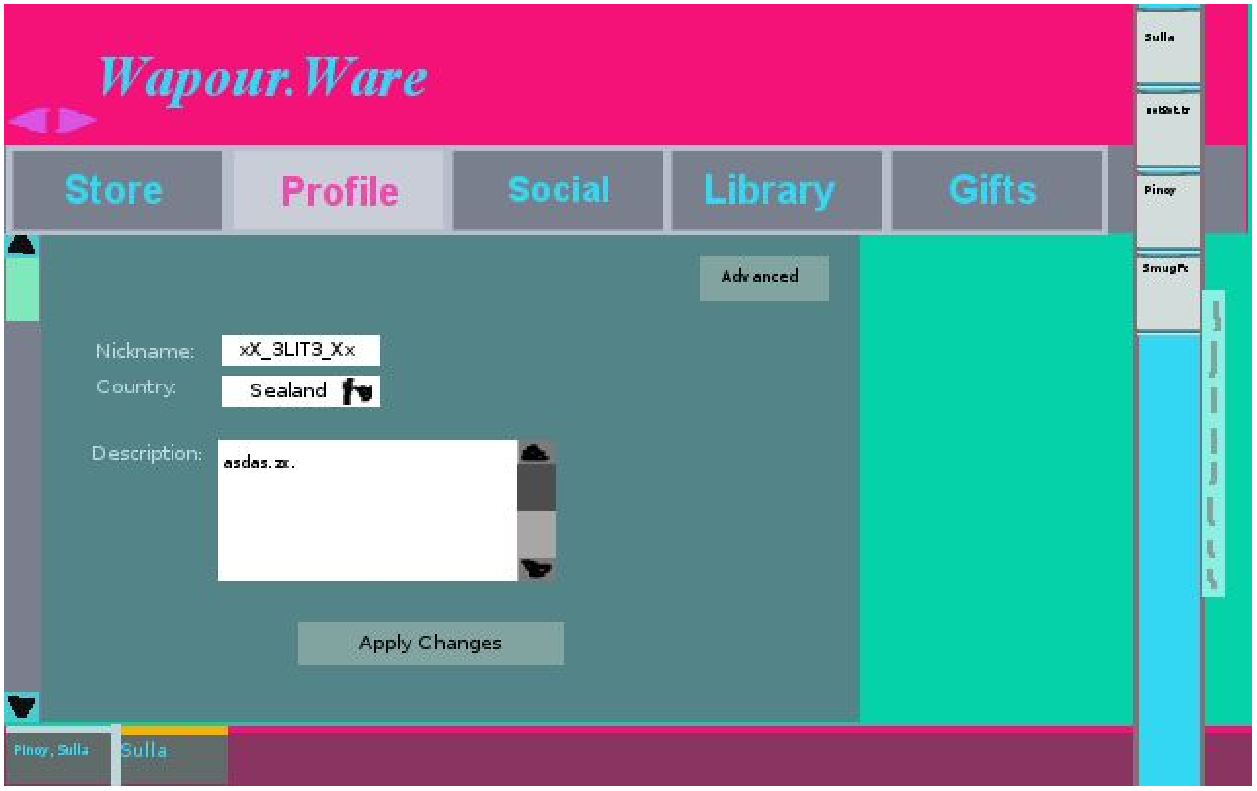


Inputs @userID. @keyword
Process: The user is looking at their groups list on the application sorted by population
SQL Statements:
SELECT Group.name,Group.ID
FROM Enrolls JOIN Group ON Enrolls.gameID = Groups.groupID
WHERE Enrolls.userID = @userID
GROUP BY GroupID
ORDER BY (COUNT(userID)) ASC
WHERE Groups.name LIKE ‘%@keyword%’



Inputs @userID, @keyword
Process: The user is looking at their friends list on the application sorted by names
SQL Statements:

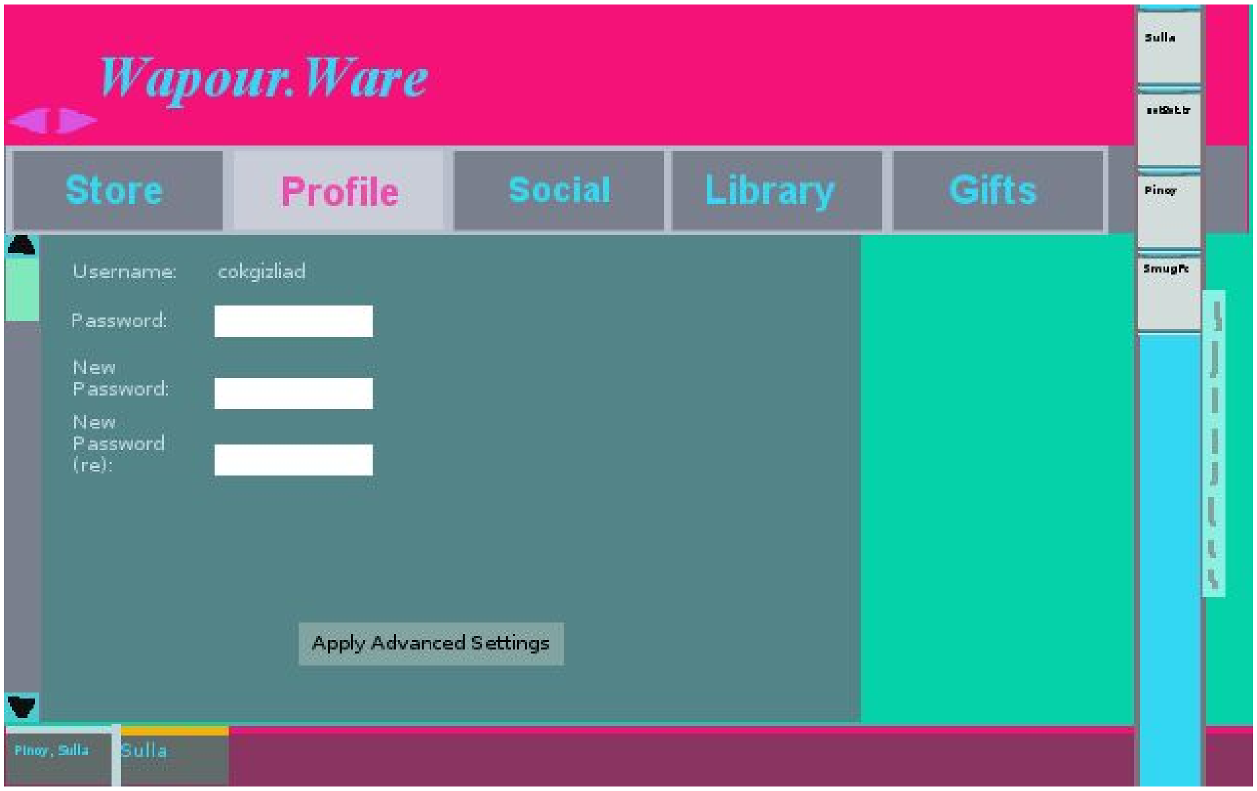
```
SELECT t2.name, t2.userID
FROM (Friends JOIN User ON Friends.user2ID = User.userID) AS t2
WHERE Friends.userID = @userID
ORDER BY t2.name ASC
WHERE t2.name LIKE '%@keyword%'
```



Nickname: xX_3LIT3_Xx
Country: Sealand
Description: asdas.zx.

Inputs @userID, @nickname, @country. @description
Process: The user is editing their profile information. Notice: nickname is not unique and can be changed, name is unique in user and cannot be changed
SQL Statements:

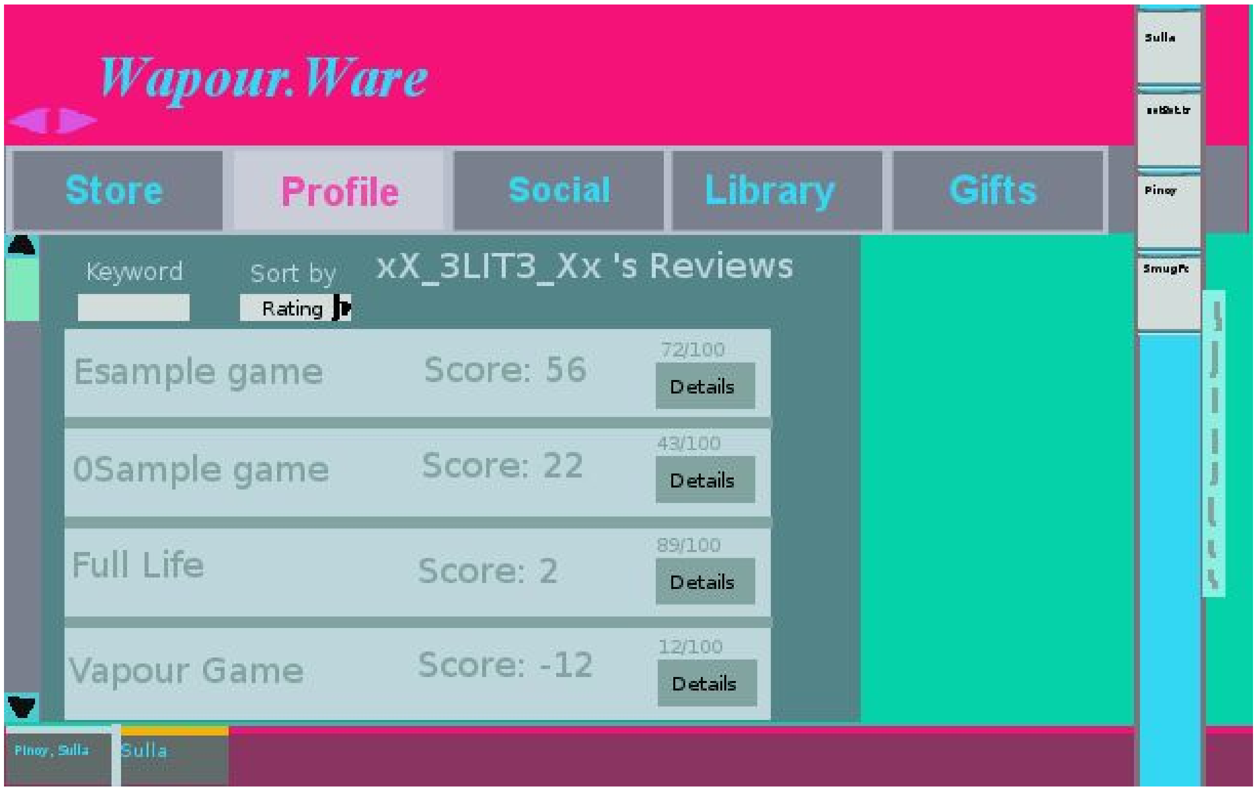
```
UPDATE User
SET nickname = @nickname, country = @country, description = @description
WHERE userID = @userID
```



Username: cokgizliad
Password:
New Password:
New Password (re):

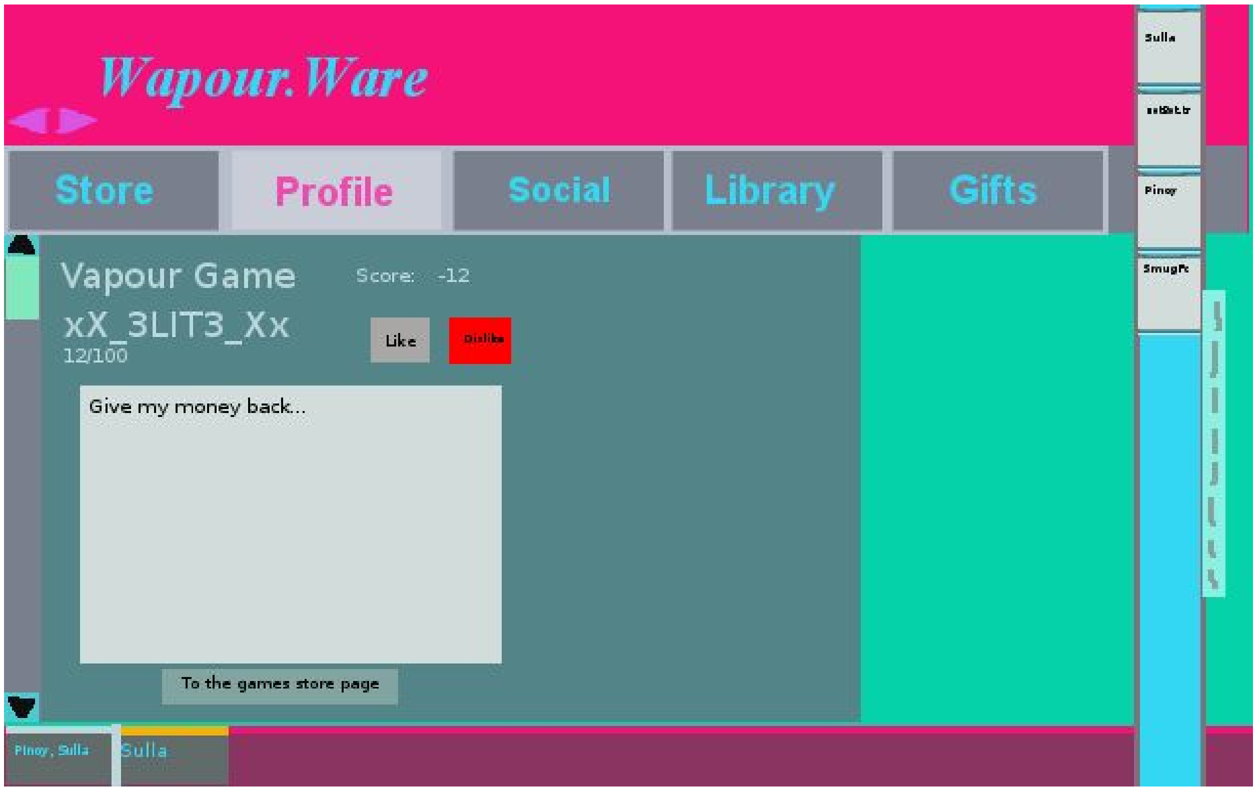
Inputs @userID, @oldpassword, @newpassword. @newpassword2
Process: The user is changing their password.
SQL Statements:

```
UPDATE User
SET password = @newpassword
WHERE password = @oldpassword AND @newpassword = @newpassword2
```



Esample game Score: 56 72/100
0Sample game Score: 22 43/100
Full Life Score: 2 89/100
Vapour Game Score: -12 12/100

Inputs @userID, @keyword
Process: The user is looking at their reviews. Notice: The rightmost side is the rating given to the game by the user. The Score part is the like minus dislikes
SQL Statements:
SELECT Game.name,Review.likes-Review.dislikes, Review.Score, Review.ID,
FROM (Reviews JOIN Writes ON Writes.ReviewID = Review.ID) AS t1 JOIN Made_To ON Made_To.ReviewID = t1.ReviewID) AS t2 JOIN Game ON Game.gameID = t2.gameID
WHERE Writes.userID = @userID
ORDER BY Review ASC
WHERE t2.name LIKE '%@keyword%'



Inputs @userID, @ReviewID, @like, @dislike

Process: The user likes or dislikes the review. Notice -12 is the result of like minus dislikes of that review. 12/100 is the rating the writer of the review gives. And a user cannot press like or dislike a game more than once.

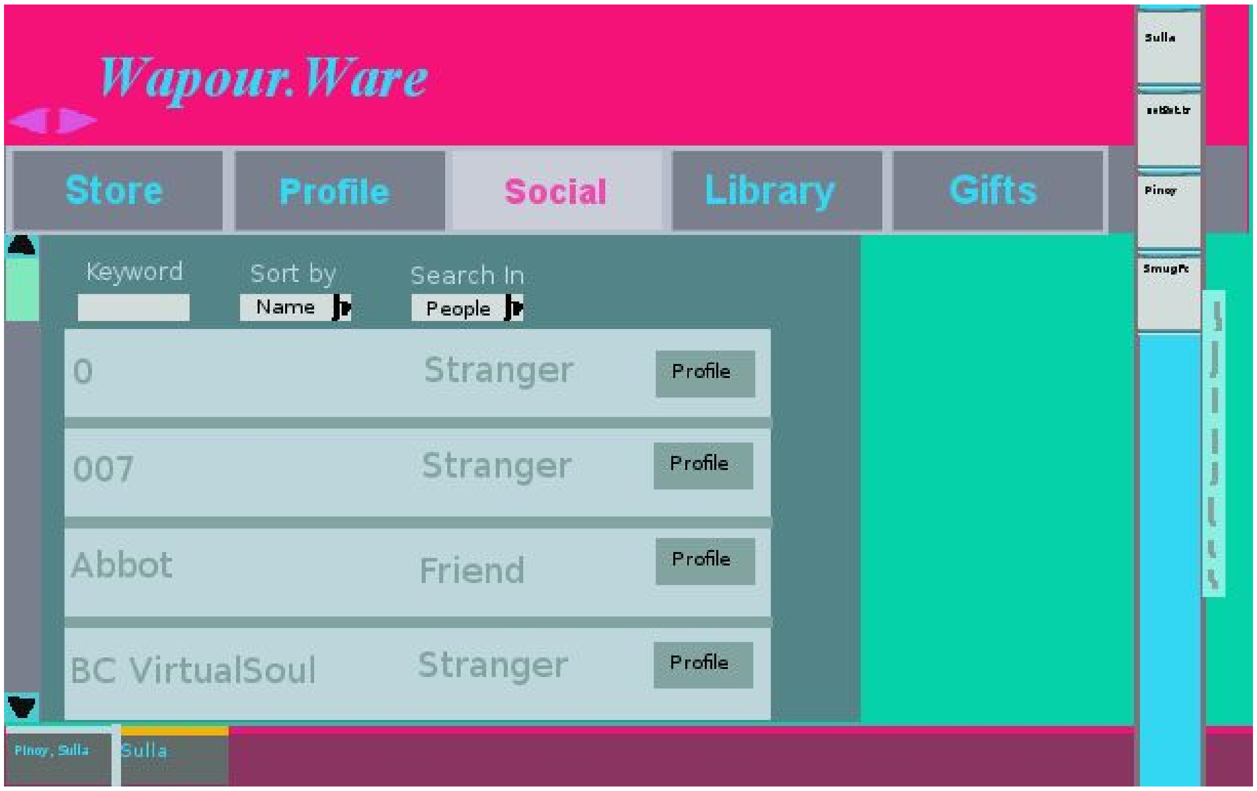
SQL Statements:

Like is Pressed

UPDATE Review
SET dislikes = dislikes-1
WHERE Tags.userID = @userID AND Tags.ReviewID = Review.ID AND R_Type = -1
UPDATE Review
SET likes = likes+1
WHERE Tags.userID = @userID AND Tags.ReviewID = Review.ID AND R_Type != 1
UPDATE Tags
SET userID = @userID, ReviewID = @ReviewID, R_type = 1

Dislike is Pressed

UPDATE Review
SET likes = likes-1
WHERE Tags.userID = @userID AND Tags.ReviewID = Review.ID AND R_Type = 1
UPDATE Review
SET dislikes = dislikes+1
WHERE Tags.userID = @userID AND Tags.ReviewID = Review.ID AND R_Type != -1
UPDATE Tags
SET userID = @userID, ReviewID = @ReviewID, R_type = -1

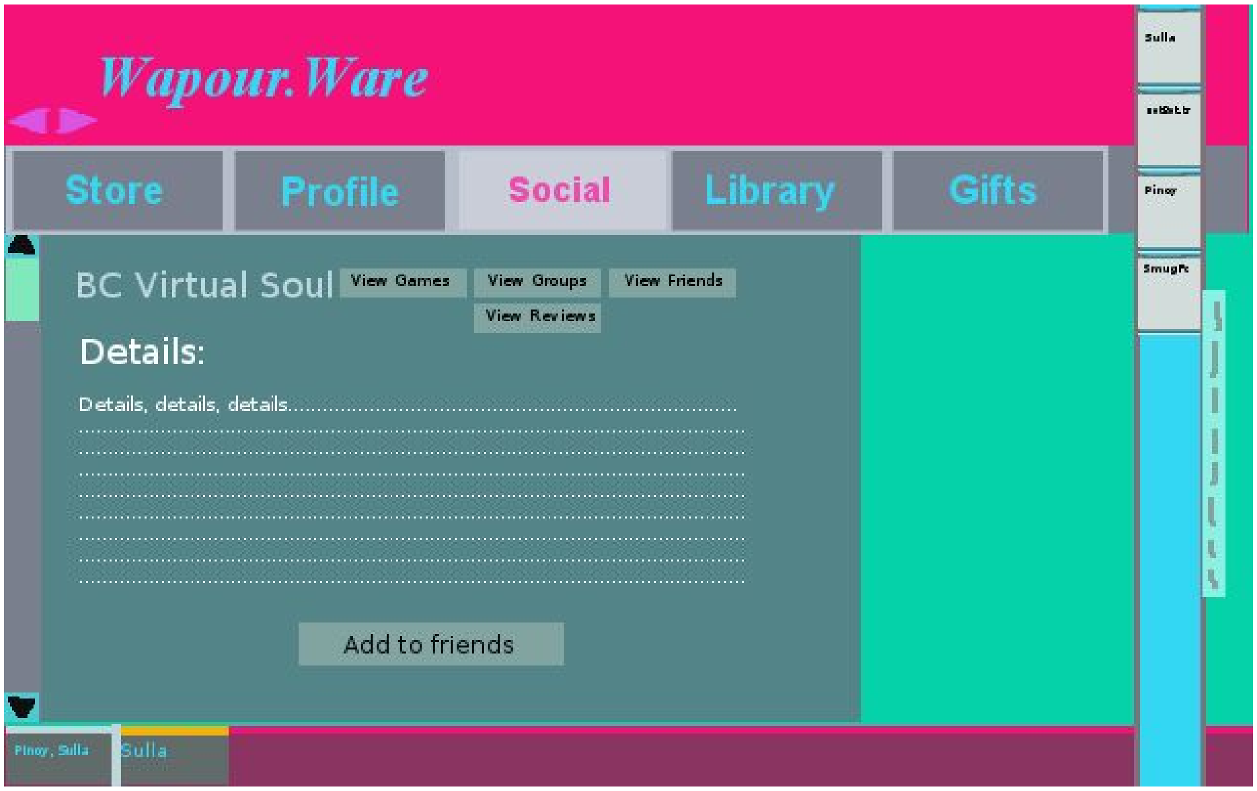


Inputs @keyword, @userID

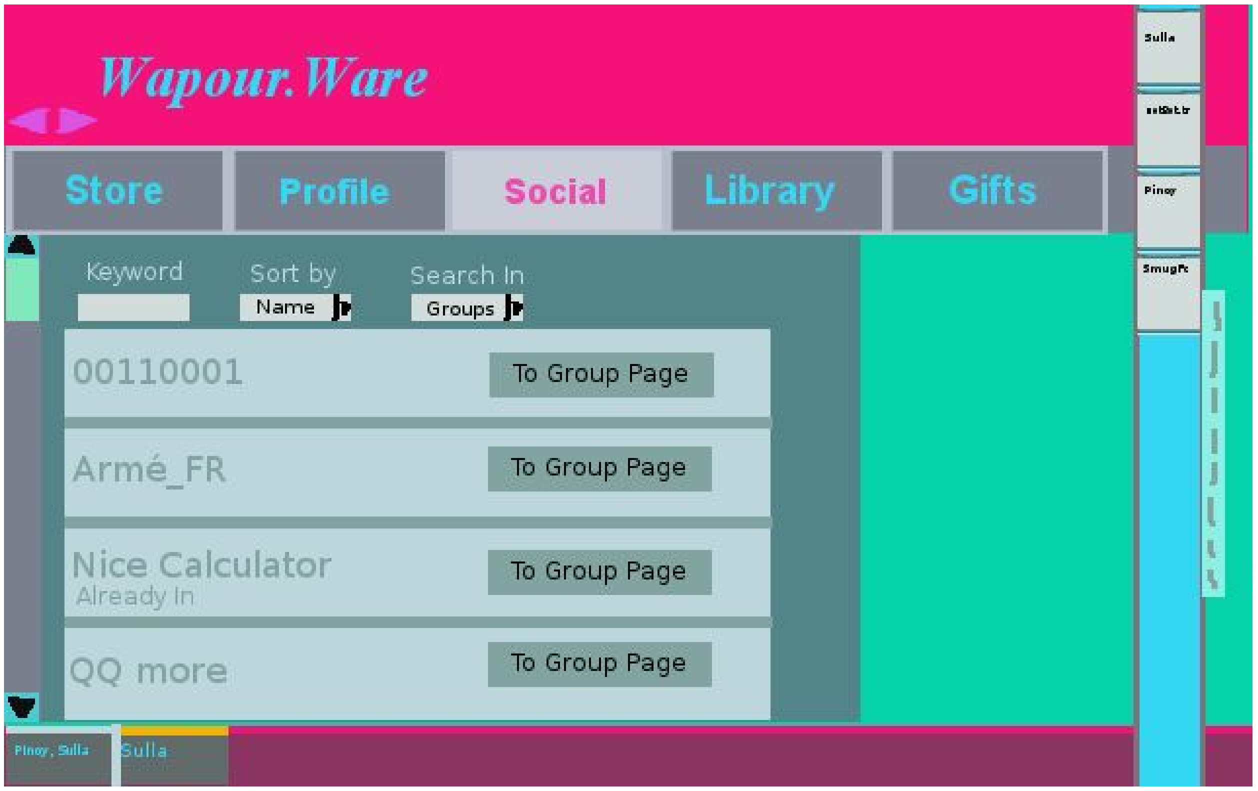
Process: The user searches for all of the users in the network.

SQL Statements:

```
SELECT t2.name, t2.groupID, (t2.userID IS NOT NULL) AS AlreadyIn
FROM (User LEFT JOIN ( SELECT * FROM Friends WHERE Friends.user2ID = @userID ) AS t1
                    ON t1.UserID = User.ID) AS t2
ORDER BY t2.name ASC
WHERE t2.name LIKE '%@keyword%'
```

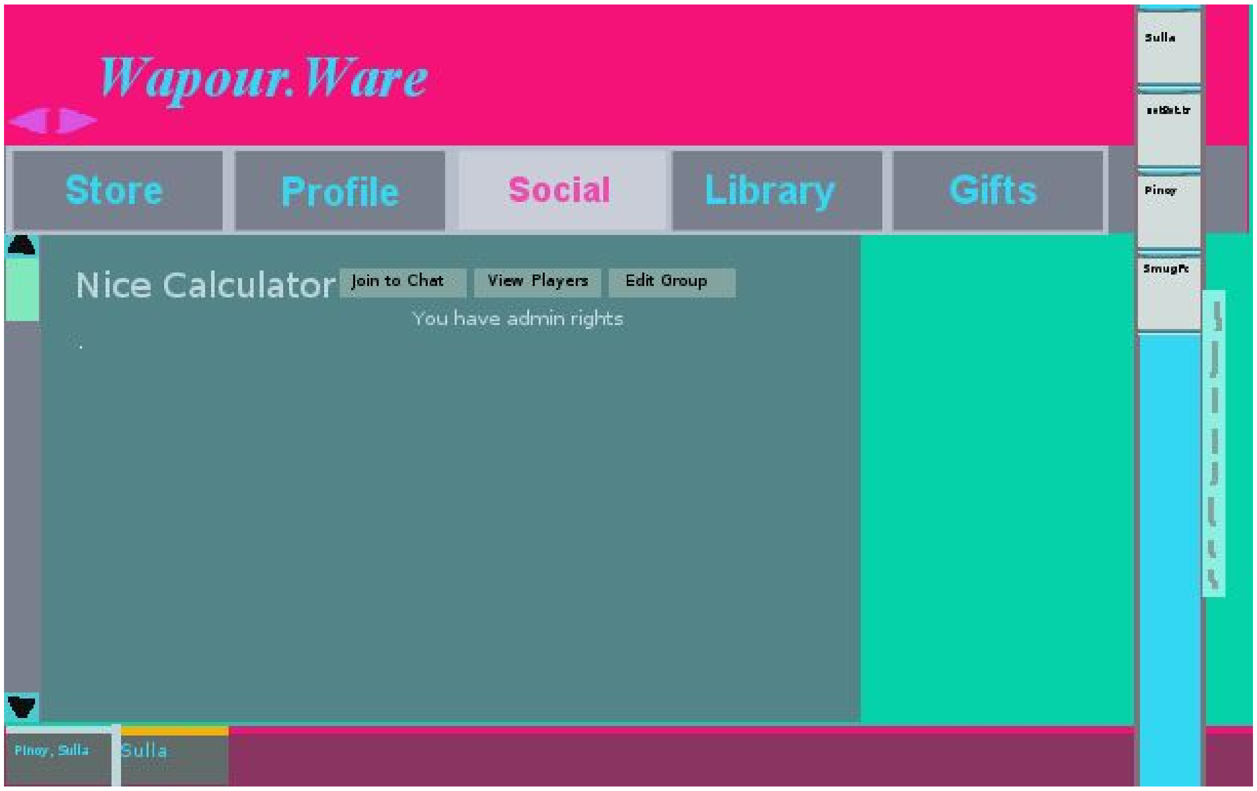


Inputs @userID, @profileID
Process: The user looks at a user profile in the network.
SQL Statements:
SELECT *
FROM User
WHERE User.ID = @profileID
Add to Friends is pressed
UPDATE Friends
SET UserID = @userID, User2ID = @profile2ID
UPDATE Friends
SET UserID = @profileID, User2ID = @userID

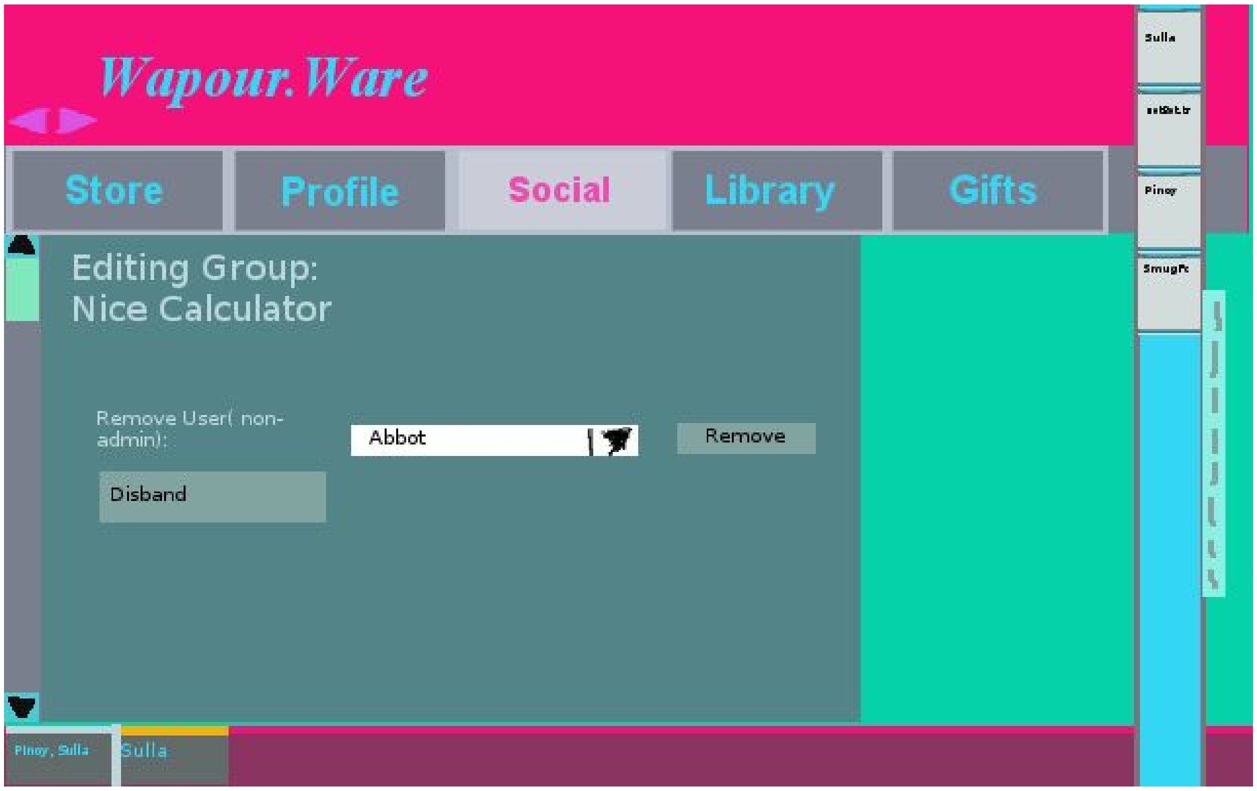


Inputs @userID, @keyword
Process: Search Groups function in the network
SQL Statements:

```
SELECT t2.name, t2.groupID, (t2.userID IS NOT NULL) AS AlreadyIn
FROM (Group LEFT JOIN ( SELECT * FROM Enrolls WHERE Enrolls.userID = @userID ) AS t1
                    ON t1.GroupID = Group.ID) AS t2
ORDER BY t2.name ASC
WHERE t2.name LIKE '%@keyword%'
```



Inputs @userID, @groupID
Process: Group Page in the network
SQL Statements:
SELECT Group.name, ChatID, (admin_ID = @userID) AS HasAdminRights
FROM Group JOIN Group_Chat
WHERE Group.ID = @groupID



Inputs @userID, @groupID, @memberID

Process: Group Page in the network

SQL Statements:

Remove is pressed

DELETE FROM Enrolls

WHERE UserID = @memberID

Disband is pressed

DELETE FROM Group

WHERE Group.ID = @groupID

DELETE FROM Enrolls

WHERE GroupID = @groupID

DELETE FROM Chat

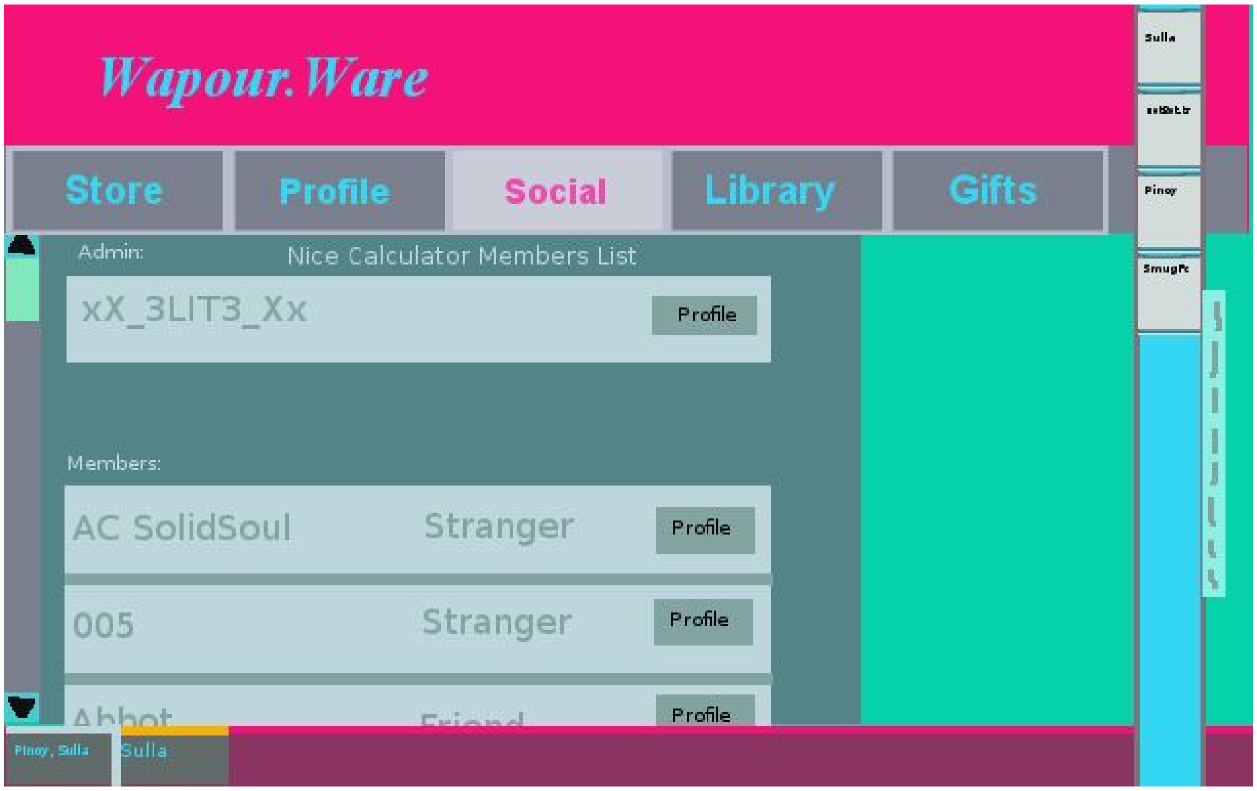
WHERE ChatID = (SELECT ChatID FROM Group_Chat WHERE GroupID= @groupID)

DELETE FROM Chats

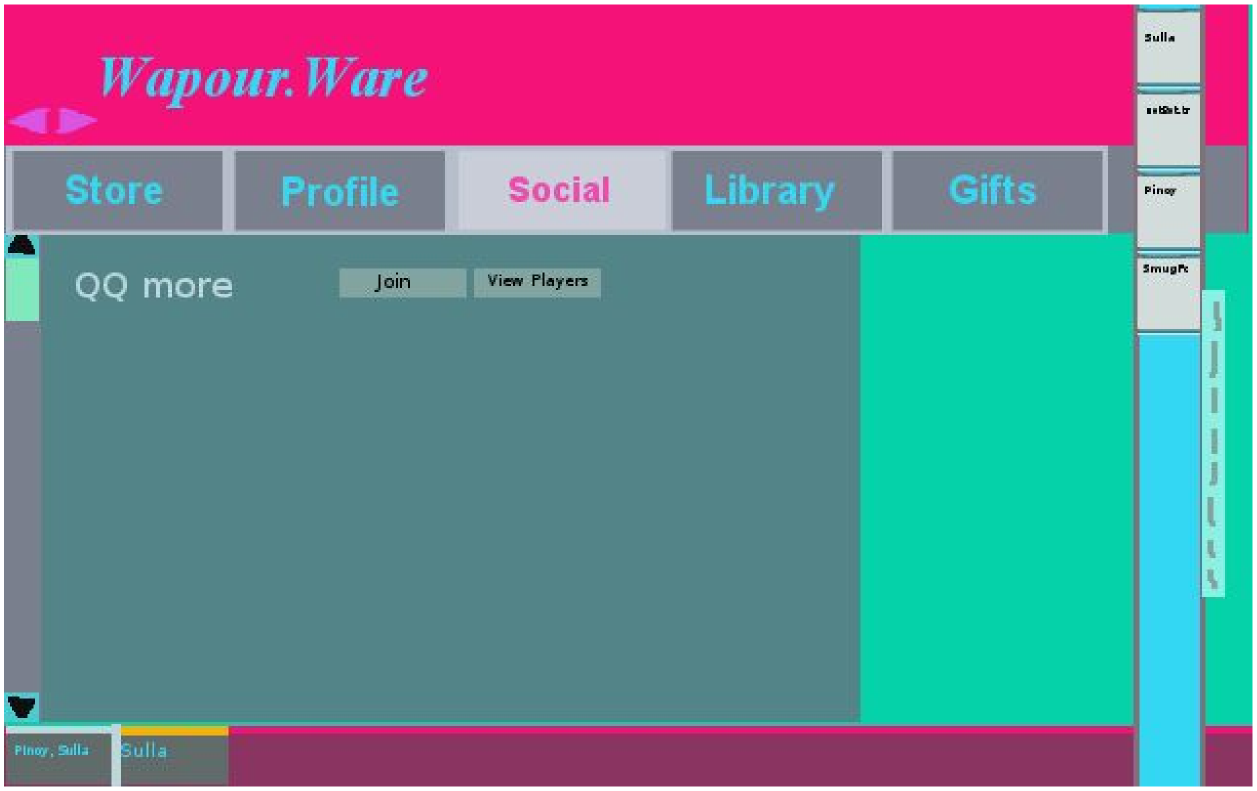
WHERE ChatID = (SELECT ChatID FROM Group_Chat WHERE GroupID= @groupID)

DELETE FROM Group_Chat

WHERE GroupID = @groupID



Inputs @userID, @groupID
Process: Group Member List in a Group
SQL Statements:
Display Admin
SELECT User.name, admin_ID
FROM Group JOIN User ON admin_ID = User.ID
WHERE Group.ID = @groupID
Display Members
SELECT User.name, UserID
FROM Enrolls JOIN User ON UserID = User.ID
WHERE GroupID = @groupID



Inputs @userID, @groupID

Process: You are in a group page you are not enrolled in

SQL Statements:

Display title

SELECT name

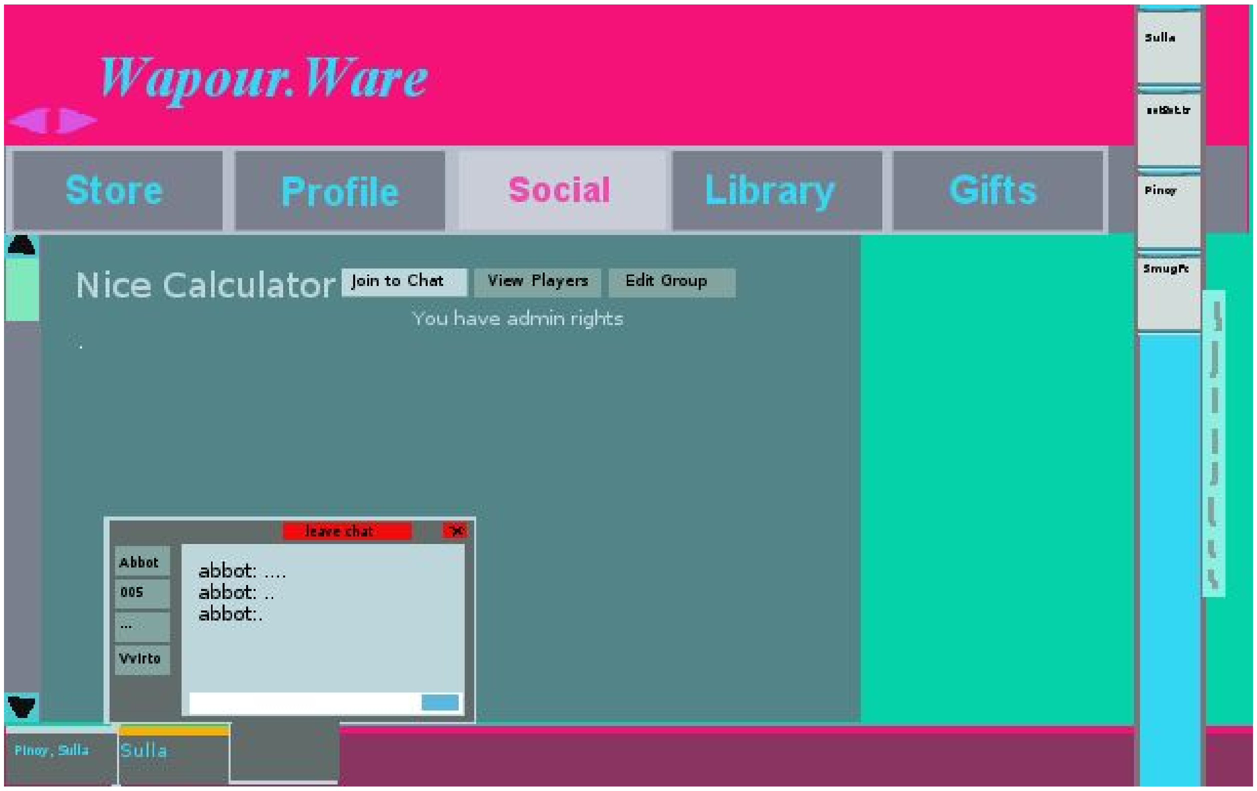
FROM Group

WHERE Group.ID = @groupID

Join is pressed

UPDATE Enrolls

SET UserID = @userID, GroupID = @groupID



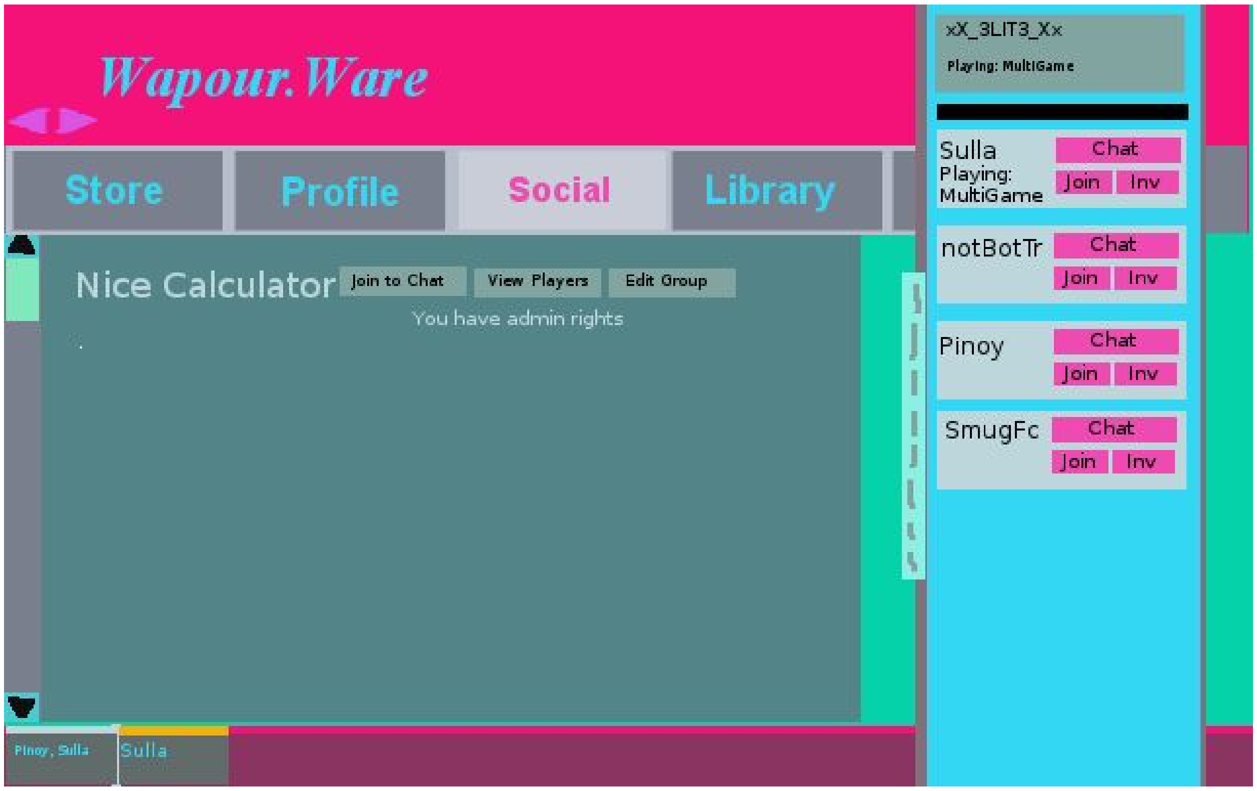
abbot: ....
abbot: ..
abbot:.

Inputs @userID, @groupID
Process: You enter the groupchat
SQL Statements:
Join to Chat

```
UPDATE Chats
SET (SELECT ChatID FROM Group_Chat WHERE GroupID = @groupID) = Chats.ChatID,
        Chats.UserID = @userID
WHERE EXISTS (SELECT GroupID, UserID
                    FROM Enrolls
                    WHERE Enrolls.groupID = @groupID AND Enrolls.UserID = @userID)
```

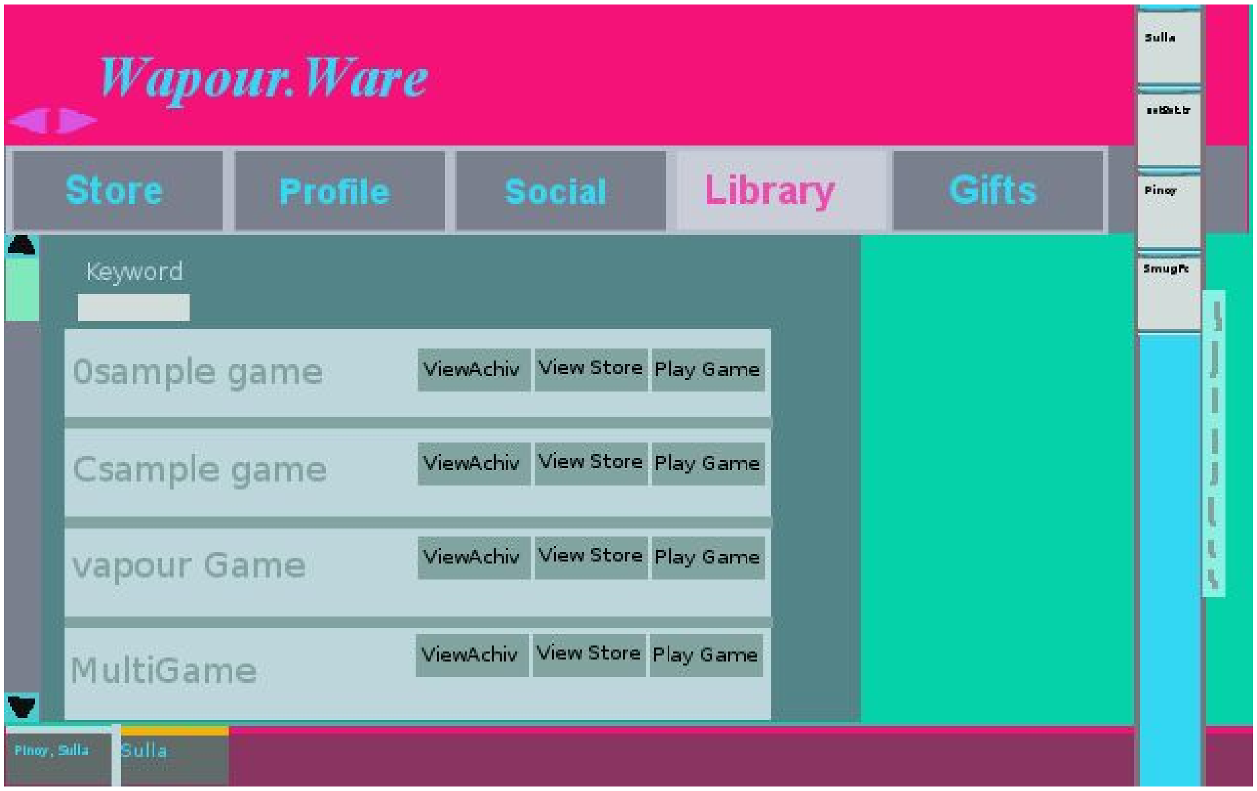
Leave Chat is Pressed

```
DELETE FROM Chats
WHERE UserID = @userID
```

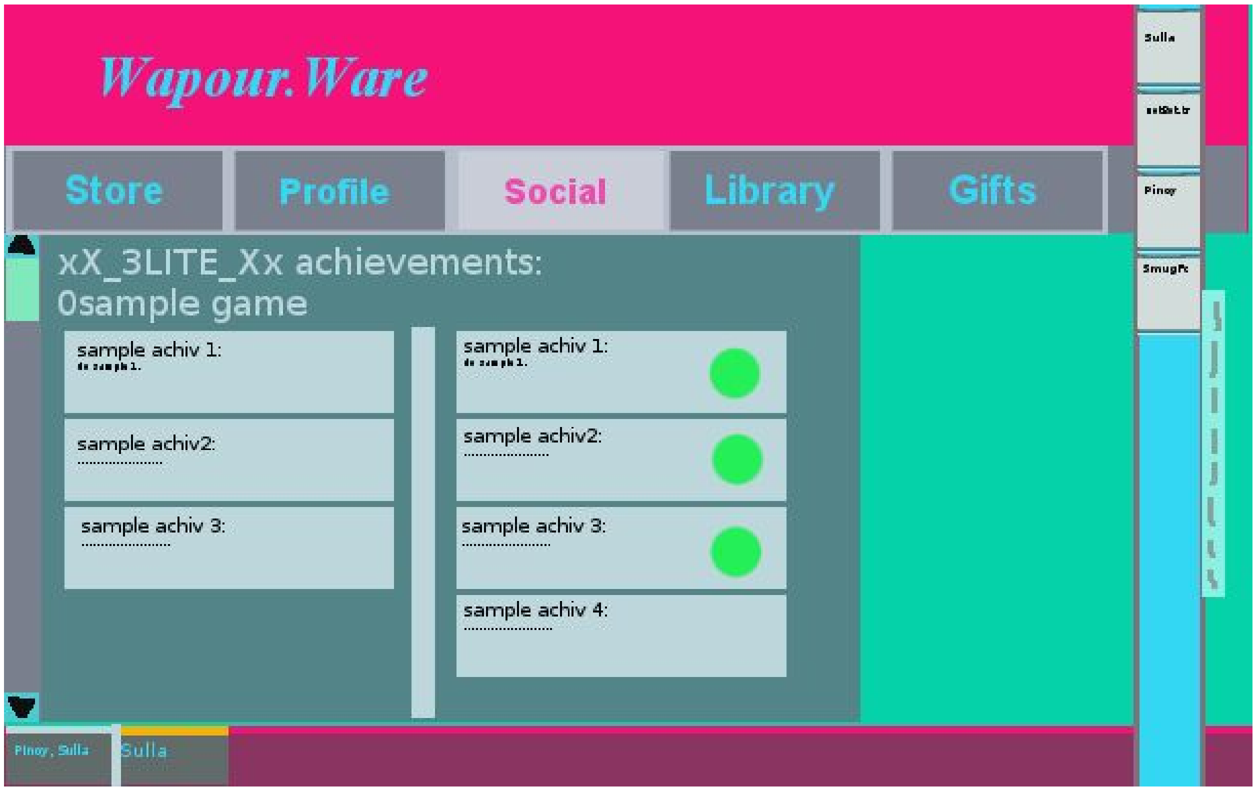


Inputs @userID, @friendID,@chatID
Process: You look at your friends list on the friends panel
SQL Statements:
Chat with a friend

```
UPDATE Chat
SET Chat.ID = @chatID
UPDATE Chats
SET Chats.UserID = @userID, Chats.ChatID = @chatID
WHERE EXISTS (SELECT UserID, User2ID
              FROM Friends
              WHERE Friends.User2ID = @friendID AND Friends.UserID = @userID)
UPDATE Chats
SET Chats.UserID = @friendID, Chats.ChatID = @chatID
WHERE EXISTS (SELECT UserID, User2ID
              FROM Friends
              WHERE Friends.User2ID = @friendID AND Friends.UserID = @userID)
```



Inputs @userID
Process: You look at your library of games
SQL Statements:
Display List of Games
SELECT Game.name, Game.ID
FROM Games JOIN Owns ON Game.ID = Owns.GameID
WHERE Owns.userID = @userID AND Game.name LIKE '%@keyword%'



Inputs @userID,@gameID

Process: You look at your achievements list belonging to a game Notice: Right side is all of the achievements, left side is the achivements the user has achived. The Green circles are present when the user has the achievement.

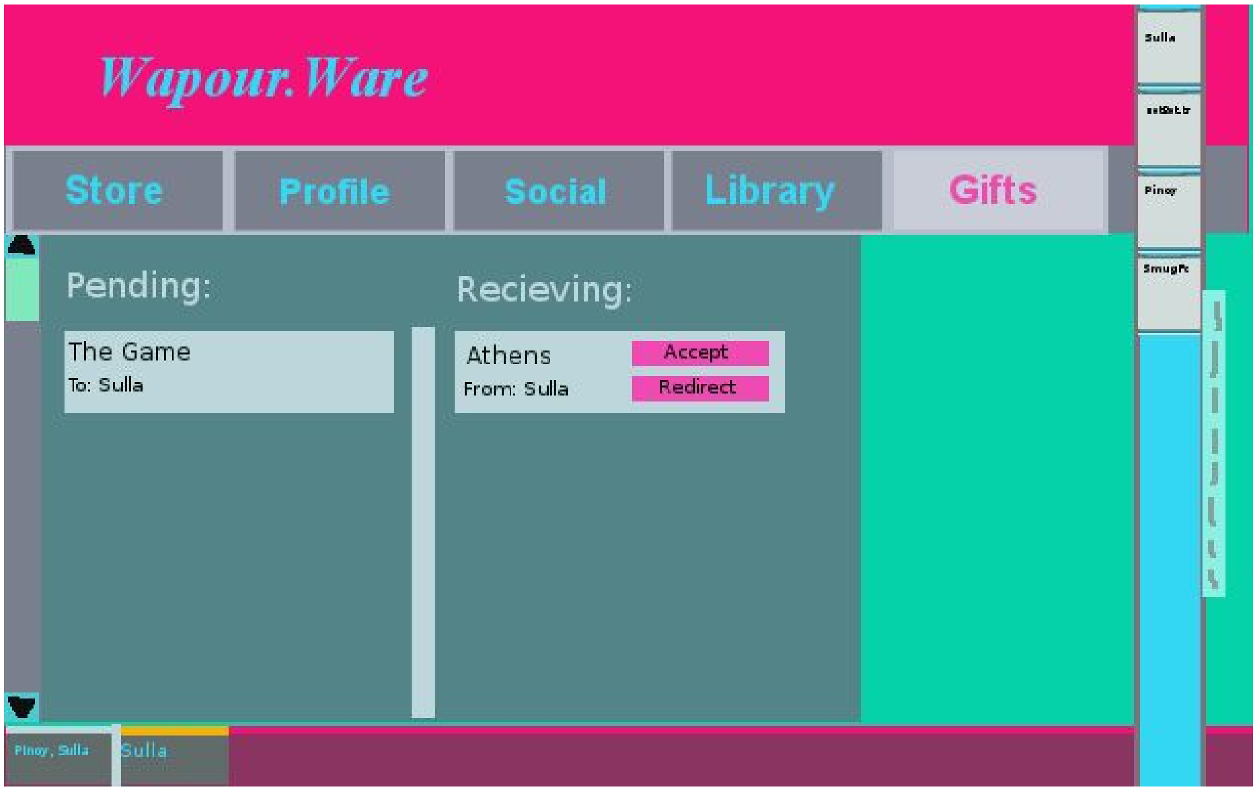
SQL Statements:

Display the list of Achievements the user has achieved

SELECT Achievement.name, Achievement.description
FROM Achievement JOIN Has ON Has.AchievementID = Achivement.ID JOIN Achieves ON Achieves.AchievementID = Achievement.ID
WHERE UserID = @userID AND GameID = @gameID

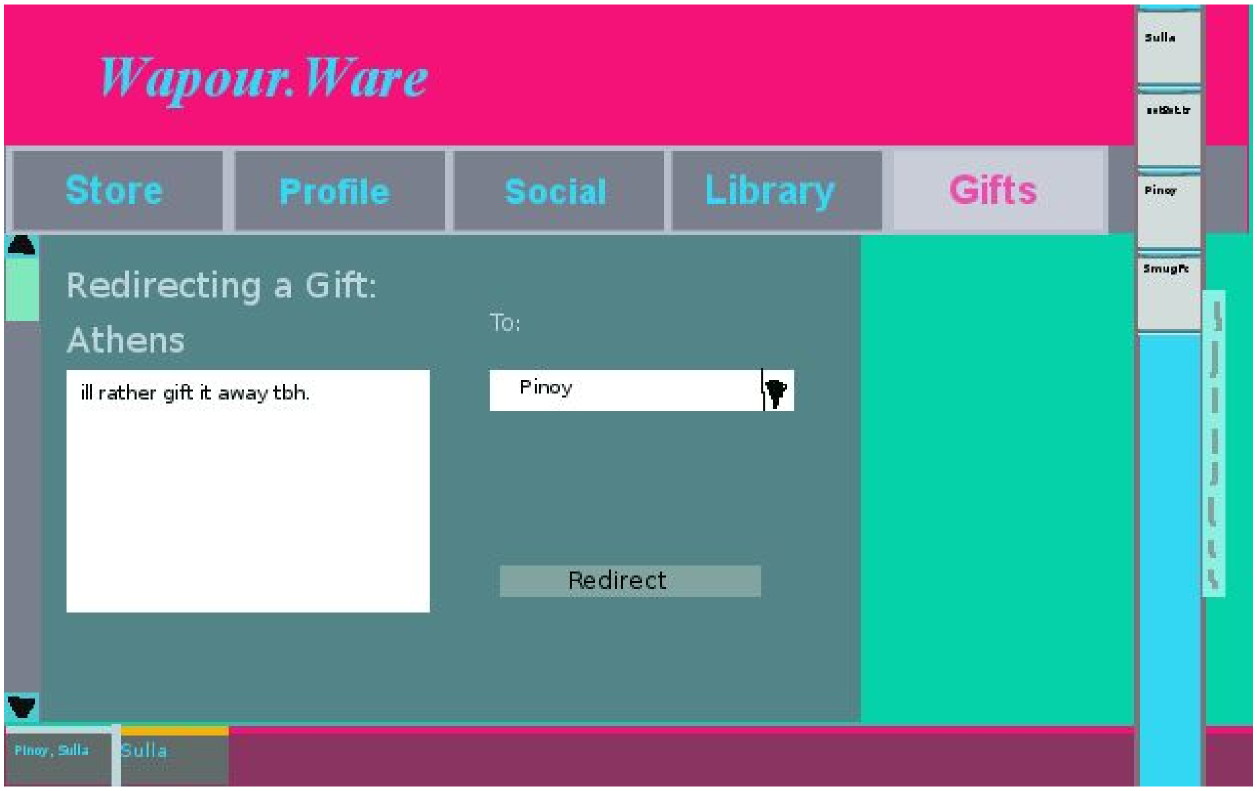
Display the list of all of the achivements in the game

SELECT Achievement.name, Achievement.description
FROM Achievement JOIN Has ON Has.AchievementID = Achivement.ID
WHERE GameID = @gameID



Inputs @userID,@giftID

Process: You look at your gifts list
SQL Statements:
Display the list of gifts you are sending
SELECT Game.name, User.name
FROM Game JOIN Contains ON Game.ID = Contain.GameID JOIN Gift ON Gift.ID = Contain.GiftID
JOIN Gift_Sends ON Gift.ID = Gift_Sends.GiftID JOIN User ON Gift_Sends.RecieverID = User.ID
WHERE Gift.ID =@giftID, Gift_Sends.SenderID = @userID
Display the list of gifts you are recieving
SELECT Game.name, User.name
FROM Game JOIN Contains ON Game.ID = Contain.GameID JOIN Gift ON Gift.ID = Contain.GiftID
JOIN Gift_Sends ON Gift.ID = Gift_Sends.GiftID JOIN User ON Gift_Sends.SenderID = User.ID
WHERE Gift.ID =@giftID, Gift_Sends.RecieverID = @userID



ill rather gift it away tbh.

Inputs @userID,@oldGiftSenderID, @newGiftReciever,@giftID, @newMessage

Process: You send your gift to another friend.
SQL Statements:
Display the list of gifts you are sending

```
DELETE FROM Gift_Sends
WHERE Gift_Sends.GiftID = @giftID AND Gift_Sends.SenderID = @oldGiftSenderID AND
        Gift_Sends.RecieverID = @userID
UPDATE Gift_Sends
SET Gift_Sends.GiftID = @giftID, Gift_Sends.SenderID = @userID,
        Gift_Sends.RecieverID = @newGiftRecieverID
UPDATE Gift
SET Gift.ID = @giftID, Gift.message = @newMessage
```

9. Stored procedures:

- Purchasing a game.
- Gaining an achievement.
- Joining into a chat.
- Joining into a group.
- Leaving a chat.

10. Triggers:

- When an administrator deletes a group,all entries defined in ‘Enrolls’ table that has the group id needs to be deleted first.

11. Constraints:

- A group can only have one administrator.
- Only members of a group can see the group chat.
- User can only join into the games of his/her friends.
- User can only invite his/her friends to games.
- Only multiplayer games can be played with friends.
- Chat invitations are not persistent, hence if not seen, they expire.
- User can only review games he owns.
- User can only play games he owns.
- Administrator can only be the creator of the group.