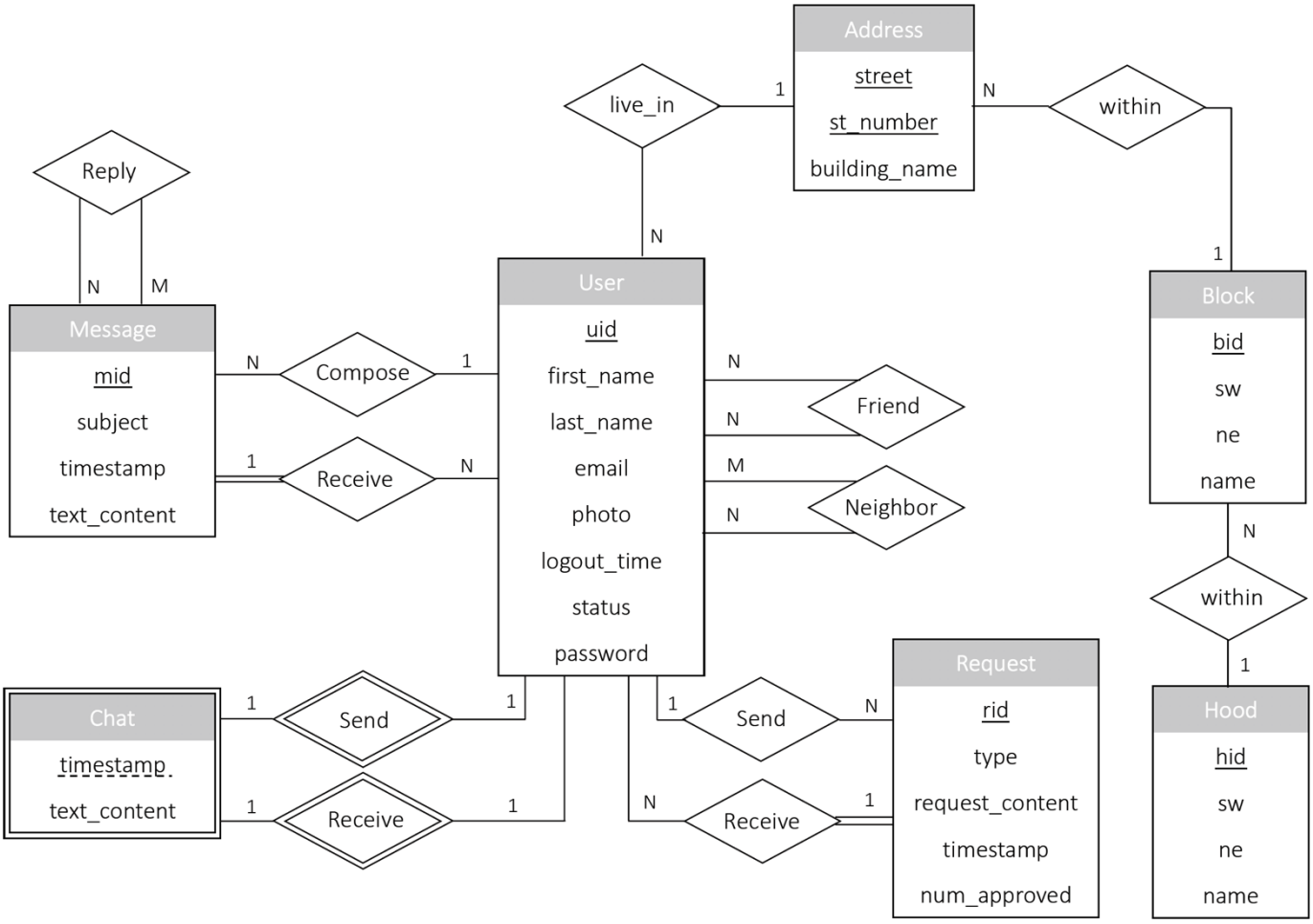
CS6083 Project #1 & #2

Yuhan Chen, Zili Xie
(yc4184, zx979)

Project #1 – Database Design

ER diagram:



Schema:

User (<u>uid</u>, first_name, last_name, email, password, photo, logout_time, status, street, st_number)
Address (<u>street</u>, <u>st_number</u>, building_name, bid, hid)
Block (<u>bid</u>, hid, sw, ne, name)
Hood (<u>hid</u>, sw, ne, name)
Request (<u>rid</u>, uid, type, request_content, timestamp, num_approved)
ReqRecipient (<u>rid</u>, <u>uid</u>)
Friend (<u>uid1</u>, <u>uid2</u>)
Neighbor (<u>uid1</u>, <u>uid2</u>)
Message (<u>mid</u>, subject, timestamp, text_content, author)
MessageRecipient (<u>mid</u>, <u>receive_uid</u>)
Reply (<u>mid_reply</u>, <u>mid_initial</u>)
Chat (<u>send_uid</u>, <u>receive_uid</u>, <u>timestamp</u>, text_content)

Key Information:

User (primary key: [uid]; foreign key: [street, st_number] references [street, st_number] from Address)
Address (primary key: [street, st_number]; foreign key: [bid] references [bid] from Block, [hid] references [hid] from Hood)
Block (primary key: [bid]; foreign key: [hid] references from [hid] Hood)
Hood (primary key: [hid])
Request (primary key: [rid]; foreign key: [uid] references [uid] from User)
ReqRecipient (primary key: [rid, uid]; foreign key: [rid] references [rid] from Request, [uid] references [uid] from User)
Friend (primary key: [uid1, uid2]; foreign key: [uid1], [uid2] both references [uid] from User)
Neighbor (primary key: [uid1, uid2]; foreign key: [uid1], [uid2] both references [uid] from User)
Message (primary key: [mid]; foreign key: [author] references [uid] from User)
MessageRecipient (primary key: [mid, uid]; foreign key: [mid] references [mid] from Message, [receive_uid] references [uid] from User)
Reply (primary key: [mid_reply, mid_initial]; foreign key: [mid_reply], [mid_initial] both references [mid] from Message)
Chat (primary key: [send_uid, receive_uid, timestamp]; foreign key: [send_uid], [receive_uid] both references [uid] from User)

Assumptions:

1. Status of a User can be ("pending", "regular", "admin"), where "pending" means that the application to join a Block for a user has not been approved by at least three members from the block. "regular" users are normal user accounts. The user(s) that has "admin" status is the administrator of a Block or Hood. Or it might be the administrator of this application.
2. Each User has an Address, where each Address is identified by a street name and number.
3. For Blocks and Hoods, sw is the coordinate of its Southwest corner. ne is the coordinate of its Northeast corner.
4. Request has two types: "friend" request and "join_block" request, where "friend" request is sent to a specific person and "join_block" request is sent to all the members in a block.
5. Message table contains initial Messages and replies. The subject attribute will be "reply" when a record is a reply.
6. An initial Message is composed by an author, sent to a group (most cases) or an individual user. Users who are in the list of recipients can read and reply to this initial message.
7. Reply messages can be associated with an initial Message. Reply to a specific message is visible to all message recipients.
8. Chats are more instant and private message between two users. User do Not need to write a "subject" or "title" for chat. Only the two users who are involved in the chat will be able to see and send this kind of messages.
9. A user can only be member of one block, and is also automatically a member of the neighborhood in which the block is located.

Tables:

```
DROP TABLE IF EXISTS USER;
CREATE TABLE USER (
    uid INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(250) NOT NULL,
    password VARCHAR(50) NOT NULL,
    photo VARCHAR(500) NOT NULL,
    logout_time DATETIME,
    status VARCHAR(20) DEFAULT 'pending',
    street VARCHAR(250) NOT NULL,
    st_number int NOT NULL,
    PRIMARY KEY (uid),
    UNIQUE (email)
);

DROP TABLE IF EXISTS BLOCK;
CREATE TABLE BLOCK(
    bid INT NOT NULL AUTO_INCREMENT,
    hid INT NOT NULL,
    name VARCHAR(250),
    sw GEOMETRY NOT NULL,
    ne GEOMETRY NOT NULL,
    PRIMARY KEY (bid),
    FOREIGN KEY (hid) REFERENCES HOOD (hid)
);


DROP TABLE IF EXISTS REQUEST;
CREATE TABLE REQUEST(
    rid INT NOT NULL AUTO_INCREMENT,
    uid INT NOT NULL,
    type VARCHAR(20) NOT NULL,
    request_content VARCHAR(5000) NOT NULL,
    timestamp DATETIME NOT NULL
                DEFAULT CURRENT_TIMESTAMP,
    num_approved INT DEFAULT 0,
    PRIMARY KEY (rid),
    FOREIGN KEY (uid) REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS HOOD;
CREATE TABLE HOOD(
    hid int NOT NULL AUTO_INCREMENT,
    name VARCHAR(250),
    sw GEOMETRY NOT NULL,
    ne GEOMETRY NOT NULL,
    PRIMARY KEY (hid)
);

DROP TABLE IF EXISTS ADDRESS;
CREATE TABLE ADDRESS(
    street VARCHAR(250) NOT NULL,
    st_number INT NOT NULL,
    building_name VARCHAR(250),
    bid INT NOT NULL,
    hid INT NOT NULL,
    PRIMARY KEY (street, st_number),
    FOREIGN KEY (bid) REFERENCES BLOCK (bid),
    FOREIGN KEY (hid) REFERENCES HOOD (hid)
);

DROP TABLE IF EXISTS REQRECIPIENT;
CREATE TABLE REQRECIPIENT(
    rid INT NOT NULL,
    uid INT NOT NULL,
    PRIMARY KEY (rid, uid),
    FOREIGN KEY (rid) REFERENCES REQUEST (rid),
    FOREIGN KEY (uid) REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS FRIEND;
CREATE TABLE FRIEND (
    uid1 INT NOT NULL,
    uid2 INT NOT NULL,
    PRIMARY KEY (uid1, uid2),
    FOREIGN KEY (uid1) REFERENCES USER (uid),
    FOREIGN KEY (uid2) REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS NEIGHBOR;
    CREATE TABLE NEIGHBOR (
    uid INT NOT NULL,
    neighbor_uid INT NOT NULL,
    PRIMARY KEY (uid, neighbor_uid),
    FOREIGN KEY (uid) REFERENCES USER (uid),
    FOREIGN KEY (neighbor_uid)
    REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS MESSAGE;
CREATE TABLE MESSAGE (
    mid INT NOT NULL AUTO_INCREMENT,
    subject VARCHAR(250) NOT NULL,
    timestamp DATETIME NOT NULL
                DEFAULT CURRENT_TIMESTAMP,
    text_content VARCHAR(5000) NOT NULL,
    author INT NOT NULL,
    PRIMARY KEY (mid),
    FOREIGN KEY (author) REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS MESSAGERECIPIENT;
CREATE TABLE MESSAGERECIPIENT(
    mid INT NOT NULL,
    receive_uid INT NOT NULL,
    PRIMARY KEY (mid, receive_uid),
    FOREIGN KEY (mid) REFERENCES MESSAGE (mid),
    FOREIGN KEY (receive_uid) REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS CHAT;
CREATE TABLE CHAT (
    send_uid INT NOT NULL,
    receive_uid INT NOT NULL,
    timestamp DATETIME NOT NULL,
    text_content VARCHAR(5000) NOT NULL,
    PRIMARY KEY (send_uid, receive_uid, timestamp),
    FOREIGN KEY (send_uid)
            REFERENCES USER (uid),
    FOREIGN KEY (receive_uid)
            REFERENCES USER (uid)
);
```

```
DROP TABLE IF EXISTS REPLY;
CREATE TABLE REPLY (
    mid_reply INT NOT NULL,
    mid_initial INT NOT NULL,
    PRIMARY KEY (mid_reply,mid_initial),
    FOREIGN KEY (mid_reply)
            REFERENCES MESSAGE (mid),
    FOREIGN KEY (mid_initial)
            REFERENCES MESSAGE (mid)
);
```

All procedures and triggers:

sign up a user:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS SIGN_UP ##
CREATE PROCEDURE SIGN_UP (IN first_name VARCHAR(50),
                          IN last_name VARCHAR(50),
                          IN email VARCHAR(250),
                          IN password VARCHAR(50),
```

```
                                        IN photo VARCHAR(500),
                                        IN street VARCHAR(250),
                                        IN st_number INT)
        BEGIN
        INSERT INTO USER (first_name, last_name, email, password, photo, street, st_number)
        VALUES (first_name, last_name,  email, password, photo, street, st_number);
        END ##
        DELIMITER ;
```

create a request:

```
        DELIMITER ##
        DROP PROCEDURE IF EXISTS OPEN_REQUEST ##
        CREATE PROCEDURE OPEN_REQUEST (IN author_uid INT,
                                       IN type VARCHAR(20),
                                       IN request_content VARCHAR(5000))
        BEGIN
        INSERT INTO REQUEST (uid, type, request_content)
        VALUES (author_uid, type, request_content);
        END ##
        DELIMITER ;
```

send request to all block members:

```
        DELIMITER ##
        DROP PROCEDURE IF EXISTS SEND_REQ_TO_BLOCK ##
        CREATE PROCEDURE SEND_REQ_TO_BLOCK (IN cur_uid INT)
        BEGIN
        SET @last_req = (
            SELECT rid
            FROM REQUEST
            WHERE REQUEST.uid = cur_uid
            Order by timestamp desc
            Limit 1
        );
        SET @cur_bid = (
            SELECT bid
            FROM USER u INNER JOIN ADDRESS a
            ON u.street = a.street and u.st_number = a.st_number
            WHERE u.uid = cur_uid
        );
        INSERT INTO REQRECIPIENT (rid, uid)
        (SELECT @last_req AS rid, u.uid AS uid
        FROM USER u INNER JOIN ADDRESS a
        ON u.street = a.street and u.st_number = a.st_number
        WHERE a.bid = @cur_bid AND u.uid <> cur_uid);
```

```
END ##
DELIMITER ;
```

Members in block approve join request:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS APPROVE_BLOCK_REQ ##
CREATE PROCEDURE APPROVE_BLOCK_REQ (IN cur_uid INT)
BEGIN
SET @last_req = (
    SELECT rid
    FROM REQUEST
    WHERE REQUEST.uid = cur_uid AND REQUEST.type = 'join_block'
    Order by timestamp desc
    Limit 1
);
UPDATE REQUEST
SET num_approved = num_approved + 1
WHERE REQUEST.rid = @last_req;
END ##
DELIMITER ;
```

Trigger that update user status when its join block request has been approved by at least three members or all members if there are less than three:

```
DELIMITER ##
DROP TRIGGER IF EXISTS MEMBER_APPROVE ##
CREATE TRIGGER MEMBER_APPROVE AFTER UPDATE ON REQUEST
FOR EACH ROW
BEGIN
SET @cur_bid = (
    SELECT bid
    FROM USER u INNER JOIN ADDRESS a
    ON u.street = a.street and u.st_number = a.st_number
    WHERE u.uid = NEW.uid
);
SET @num_member = (
    SELECT COUNT(*) FROM
    USER u INNER JOIN ADDRESS a
    ON u.street = a.street and u.st_number = a.st_number
    WHERE a.bid = @cur_bid AND (u.status = 'regular' OR u.status = 'admin')
);
IF (NEW.num_approved = 3 OR NEW.num_approved = @num_member)
    AND NEW.type = 'join_block' THEN
UPDATE USER
SET status = 'regular'
```

```
WHERE uid = NEW.uid;
END IF;
END ##
DELIMITER ;
```

The user update their address:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS UPDATE_ADDRESS ##
CREATE PROCEDURE UPDATE_ADDRESS (IN new_street VARCHAR(250),
                                 IN new_st_number INT,
                                 IN cur_uid INT)
BEGIN
UPDATE USER
SET  USER.street = new_street,
     USER.st_number = new_st_number,
     USER.status = 'pending'
   where  USER.uid = cur_uid;
END ##
DELIMITER ;
```

The user update their email:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS UPDATE_EMAIL ##
CREATE PROCEDURE UPDATE_EMAIL (IN new_email VARCHAR(250), IN cur_uid INT)
BEGIN
UPDATE USER
SET USER.email = new_email
WHERE USER.uid = cur_uid;
END ##
DELIMITER ;
```

The user update their photo:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS UPDATE_PHOTO ##
CREATE PROCEDURE UPDATE_PHOTO (IN cur_uid INT, IN new_photo VARCHAR(500))
BEGIN
UPDATE USER
SET USER.photo = new_photo
WHERE USER.uid = cur_uid;
END ##
DELIMITER ;
```

User compose an initial message:

```
DELIMITER ##
```

```
DROP PROCEDURE IF EXISTS COMPOSE_MESSAGE ##
CREATE PROCEDURE COMPOSE_MESSAGE (
        IN subject VARCHAR(250),
        IN text_content VARCHAR(5000),
        IN uid INT)
BEGIN
INSERT INTO MESSAGE (subject, text_content, author)
VALUES (subject, text_content, uid
);
END ##
DELIMITER ;
```

User assign recipient to an initial message:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS DIRECT_MESSAGE ##
CREATE PROCEDURE DIRECT_MESSAGE (IN uid INT, IN receive_uid INT)
BEGIN
SET @last_msg = (
    SELECT mid
    FROM MESSAGE
    WHERE author = uid
    Order by timestamp desc
    Limit 1
    );
    INSERT INTO MESSAGERECIPIENT VALUES (@last_msg, receive_uid);
END ##
DELIMITER ;
```

User reply to a message:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS REPLY_MESSAGE ##
CREATE PROCEDURE REPLY_MESSAGE (IN uid INT, IN mid_initial INT)
BEGIN
SET @last_msg = (
    SELECT mid
    FROM MESSAGE
    WHERE author = uid
    Order by timestamp desc
    Limit 1
);
INSERT INTO REPLY VALUES (@last_msg, mid_initial);
END ##
DELIMITER ;
```

Send friend request:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS SEND_REQ_TO_MEM ##
CREATE PROCEDURE SEND_REQ_TO_MEM (IN cur_uid INT, IN receive_uid INT)
BEGIN
SET @last_req = (
SELECT rid
FROM REQUEST
WHERE REQUEST.uid = cur_uid
Order by timestamp desc
Limit 1
);
INSERT INTO REQRECIPIENT (rid, uid)
VALUES (@last_req, receive_uid);
END ##
DELIMITER ;
```

Approve friend request:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS APPROVE_FRIEND_REQ ##
CREATE PROCEDURE APPROVE_FRIEND_REQ (IN cur_uid INT, IN my_uid INT)
BEGIN
SET @last_req = (
SELECT REQUEST.rid
FROM REQUEST INNER JOIN REQRECIPIENT ON REQUEST.rid = REQRECIPIENT.rid
WHERE REQUEST.uid = cur_uid AND REQUEST.type = 'friend' AND REQRECIPIENT.uid = my_uid
);
UPDATE REQUEST
SET num_approved = num_approved + 1
WHERE REQUEST.rid = @last_req;
END ##
DELIMITER ;
```

Trigger that insert record to Friend once the friend request has been approved.

```
DELIMITER ##
DROP TRIGGER IF EXISTS FRIEND_APPROVE ##
CREATE TRIGGER FRIEND_APPROVE AFTER UPDATE ON REQUEST
FOR EACH ROW
BEGIN
SET @receive_uid = (
    SELECT REQRECIPIENT.uid FROM
    REQUEST INNER JOIN REQRECIPIENT ON REQUEST.rid = REQRECIPIENT.rid
    WHERE REQUEST.type = 'friend' AND REQUEST.rid = NEW.rid
);
```

```
IF NEW.num_approved = 1 AND NEW.type = 'friend' THEN
INSERT INTO FRIEND (uid1, uid2) VALUES (NEW.uid, @receive_uid);
END IF;
END ##
DELIMITER ;
```

List all current friends:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS LIST_FRIEND ##
CREATE PROCEDURE LIST_FRIEND (IN cur_uid INT)
BEGIN
DROP TEMPORARY TABLE IF EXISTS temp;
CREATE TEMPORARY TABLE temp
WITH FRIEND_UIDS AS (
    (SELECT uid2 AS friend_uid
    FROM FRIEND
    WHERE uid1 = cur_uid)
    UNION
    (SELECT uid1 AS friend_uid
    FROM FRIEND
    WHERE uid2 = cur_uid)
)
SELECT u.uid, u.first_name, u.last_name
FROM FRIEND_UIDS fu INNER JOIN USER u
ON fu.friend_uid = u.uid;
END ##
DELIMITER ;
```

Add someone as neighbor:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS ADD_NEIGHBOR ##
CREATE PROCEDURE ADD_NEIGHBOR (IN cur_uid INT, IN neighbor_uid INT)
BEGIN
INSERT INTO NEIGHBOR (uid, neighbor_uid) VALUES (cur_uid, neighbor_uid);
END ##
DELIMITER ;
```

List all current neighbors:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS LIST_NEIGHBOR ##
CREATE PROCEDURE LIST_NEIGHBOR (IN cur_uid INT)
BEGIN
DROP TEMPORARY TABLE IF EXISTS temp;
CREATE TEMPORARY TABLE temp;
```

```
WITH NEIGHBOR_UIDS AS (
    SELECT neighbor_uid AS neighbor_uid
    FROM NEIGHBOR
    WHERE uid = cur_uid
)
SELECT u.uid, u.first_name, u.last_name
FROM NEIGHBOR_UIDS nu INNER JOIN USER u
ON nu.neighbor_uid = u.uid;
END ##
DELIMITER ;
```

List all threads in a user's block feed:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS BLOCK_FEED ##
CREATE PROCEDURE BLOCK_FEED (IN cur_uid INT)
BEGIN
SET @cur_bid = (
    SELECT bid
    FROM USER u INNER JOIN ADDRESS a
    ON u.street = a.street and u.st_number = a.st_number
    WHERE u.uid = cur_uid
);
SELECT MESSAGE.mid, MESSAGE.subject, MESSAGE.text_content
FROM MESSAGERECIPIENT INNER JOIN
    MESSAGE ON MESSAGERECIPIENT.mid = MESSAGE.mid
    INNER JOIN USER ON MESSAGE.author = USER.uid
    INNER JOIN ADDRESS ON USER.street = ADDRESS.street AND USER.st_number =
ADDRESS.st_number
WHERE MESSAGERECIPIENT.receive_uid = cur_uid AND ADDRESS.bid = @cur_bid AND
MESSAGE.timestamp > USER.logout_time;
END ##
DELIMITER ;
```

List all threads in friends feed;

```
DELIMITER ##
DROP PROCEDURE IF EXISTS FRIEND_FEED ##
CREATE PROCEDURE FRIEND_FEED (IN cur_uid INT)
BEGIN
CALL LIST_FRIEND(cur_uid);
SET @last_logout_time = (
SELECT logout_time
FROM USER
WHERE USER.uid = cur_uid
);
```

```
DROP TEMPORARY TABLE IF EXISTS temp2;
CREATE TEMPORARY TABLE temp2
SELECT MESSAGE.*
FROM MESSAGERECIPIENT INNER JOIN
MESSAGE ON MESSAGERECIPIENT.mid = MESSAGE.mid
INNER JOIN temp ON MESSAGE.author = temp.uid
WHERE MESSAGERECIPIENT.receive_uid = cur_uid
AND MESSAGE.timestamp > @last_logout_time;
END ##
DELIMITER ;
```

Search message that are visible to user:

```
DELIMITER ##
DROP PROCEDURE IF EXISTS SEARCH_MESSAGE ##
CREATE PROCEDURE SEARCH_MESSAGE (IN cur_uid INT, IN keyword VARCHAR(50))
BEGIN
DROP TEMPORARY TABLE IF EXISTS temp;
CREATE TEMPORARY TABLE temp
SELECT MESSAGE.*
FROM MESSAGERECIPIENT INNER JOIN
MESSAGE ON MESSAGERECIPIENT.mid = MESSAGE.mid
WHERE MESSAGERECIPIENT.receive_uid = cur_uid AND (MESSAGE.subject LIKE CONCAT('%',
keyword, '%') OR MESSAGE.text_content LIKE CONCAT('%', keyword, '%'));
END ##
DELIMITER ;
```

Project #2 – Function realization and UI design

(1) Sign up user:

Sign up a user with Password Encryption:

```
CALL SIGN_UP ('{$_POST['first_name']}', '{$_POST['last_name']}',  '{$_POST['email']}',
                '{$_POST['password']}',  '{$_POST['photo_path']}',
                '{$_POST['street']}', {$_POST['st_number']});
```

In PHP we use MD5 encryption to make the password secure in Database :

```php
function sign_up_user() {
    $servername = "127.0.0.1";
    $username = "root";
    $password = "root";
    $dbname = "neighborhood";
    $conn = new mysqli($servername, $username, $password, $dbname);

    $first_spc = strpos($_POST['address'], ' ');
    $no = substr($_POST['address'], 0, $first_spc);
    $str = substr($_POST['address'], $first_spc + 1);

    $sql = "CALL SIGN_UP ('{$_POST['fn']}', '{$_POST['ln']}',
                '{$_POST['email']}', MD5('{$_POST['password']}'), 'C:/url', '{$str}', {$no});";
```

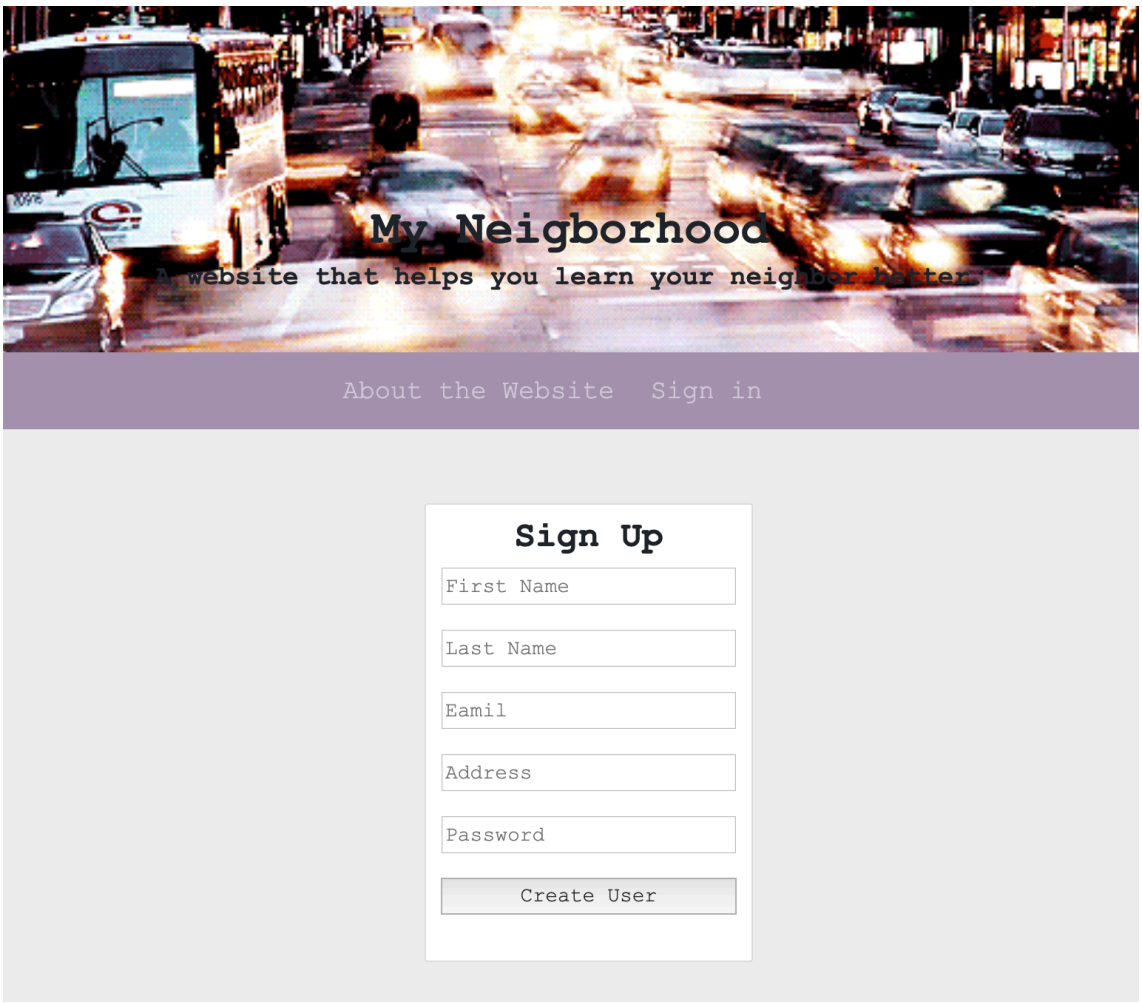
```php
    $signup = $conn->query($sql);

    if ($signup) {
        $_SESSION['current_user_email'] = $_POST['email'];

        // echo "<p>$str</p>";
        // echo "<p>$no</p>";
        echo "<p> sign up succeed </p>";
        $request = $conn->query("CALL OPEN_REQUEST({$uid}, 'join_block', '{$_POST['req_content']}');");

        $_SESSION['current_user_name'] = ucfirst($_POST['fn'])." ".ucfirst($_POST['ln']);
        $_SESSION['current_address'] = $_POST['address'];
        $_SESSION['current_status'] = 'pending';
        echo '<script>window.location.href = "home.php";</script>';
    } else {
        echo "<p style='color:red;'> sign up failed </p>";
    }
}
```

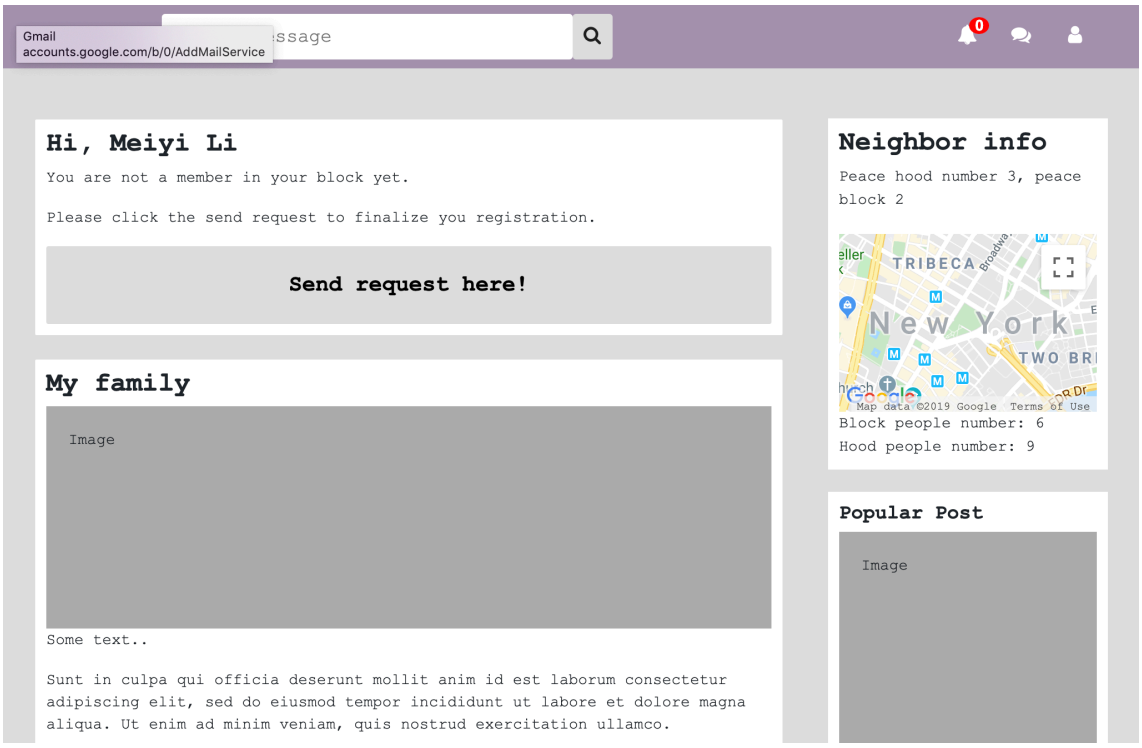
Our User Interface shown as below. User needs to provide their First and Last name, Email which enforced to be identical, address and password. The storage of password will be encrypted in Database.



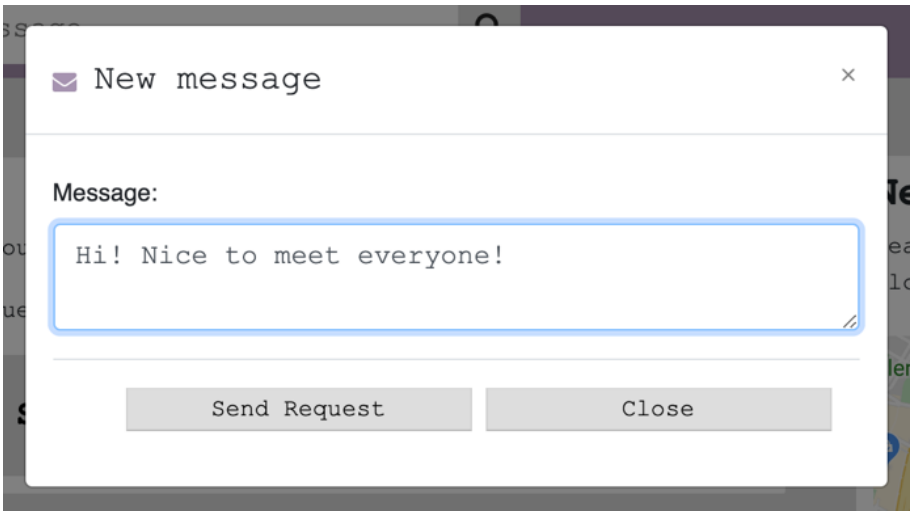
The input of address can be autocompleted. This is done by searching all the matching record from backend given user input.

(2) New User Home page (pending status):

A newly signed up user can see the interface below. New user needs to send request to join the current block the have access to the complete functions of this website.



Clicking the send request button will pop up a Window that asks for greeting message. This request will be send to the other members in this block.



Sending Request is done by calling the OPEN_REQUEST procedure which insert a request to the REQUEST table and SEND_REQ_TO_BLOCK procedure, which directs the request to target recipients.

CALL OPEN_REQUEST({$uid}, 'join_block', '{$_POST['req_content']}');
CALL SEND_REQ_TO_BLOCK ({$uid})

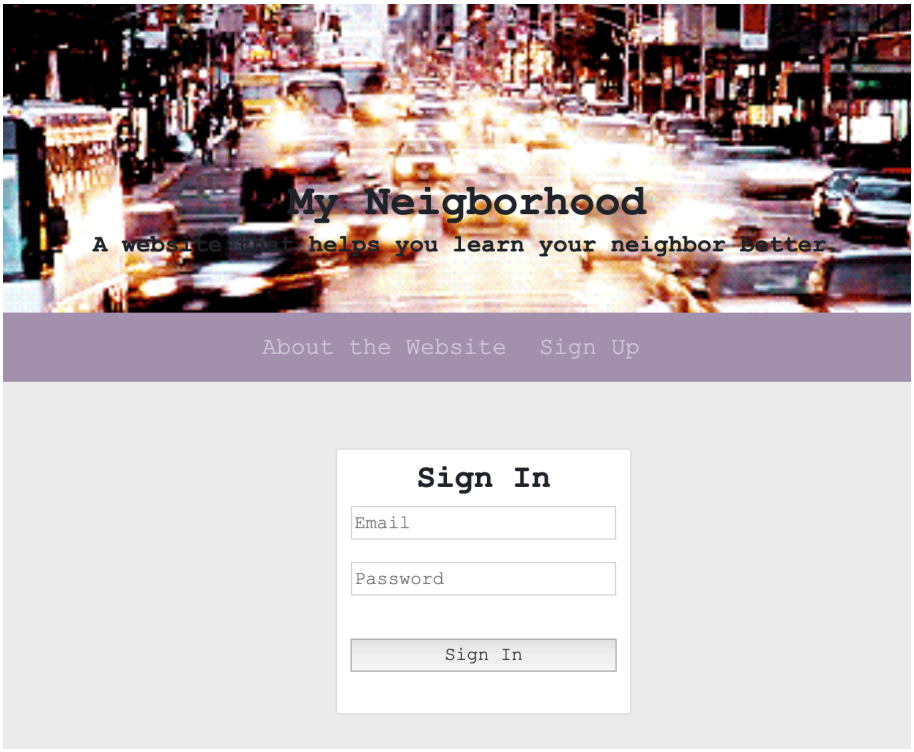
In PHP:

```php
function send_req() {
    $servername = "127.0.0.1";
    $username = "root";
    $password = "root";
    $dbname = "neighborhood";
    $conn = new mysqli($servername, $username, $password, $dbname);

    $sql = "CALL OPEN_REQUEST({$_SESSION['current_uid']}, 'join_block', '{$_POST['message-text']}');";
    $sendreq = $conn->query($sql);
    if ($sendreq) {
        $sendreq2block = $conn->query("CALL SEND_REQ_TO_BLOCK ({$_SESSION['current_uid']})");
        if ($sendreq2block) {
            echo "<p>request has been send.</p>";
            echo "<p>0/3 approvals have been collected.</p>";
        }
    }
}
```

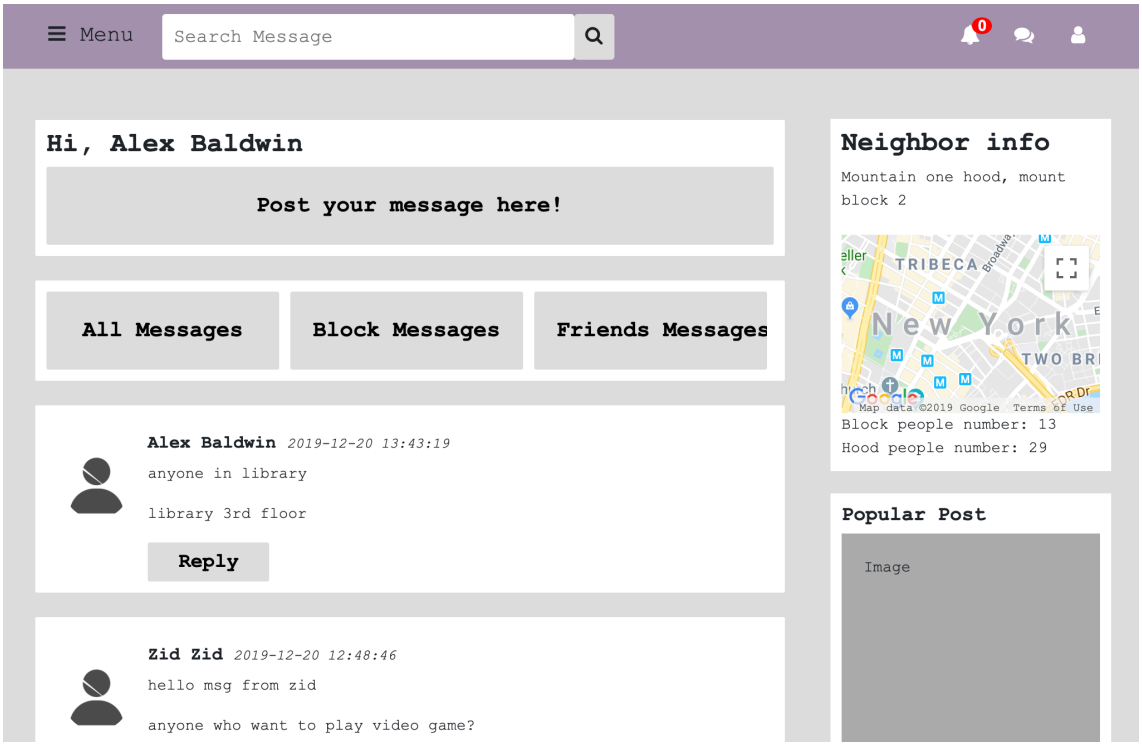
(3) Sign in user:

User can also choose to sign in with existing account by clicking the "Sign In" button in the Sign up page. It will direct the website to Signin.php. The User Interface is shown as below. Existing user can sign in with their email and password.

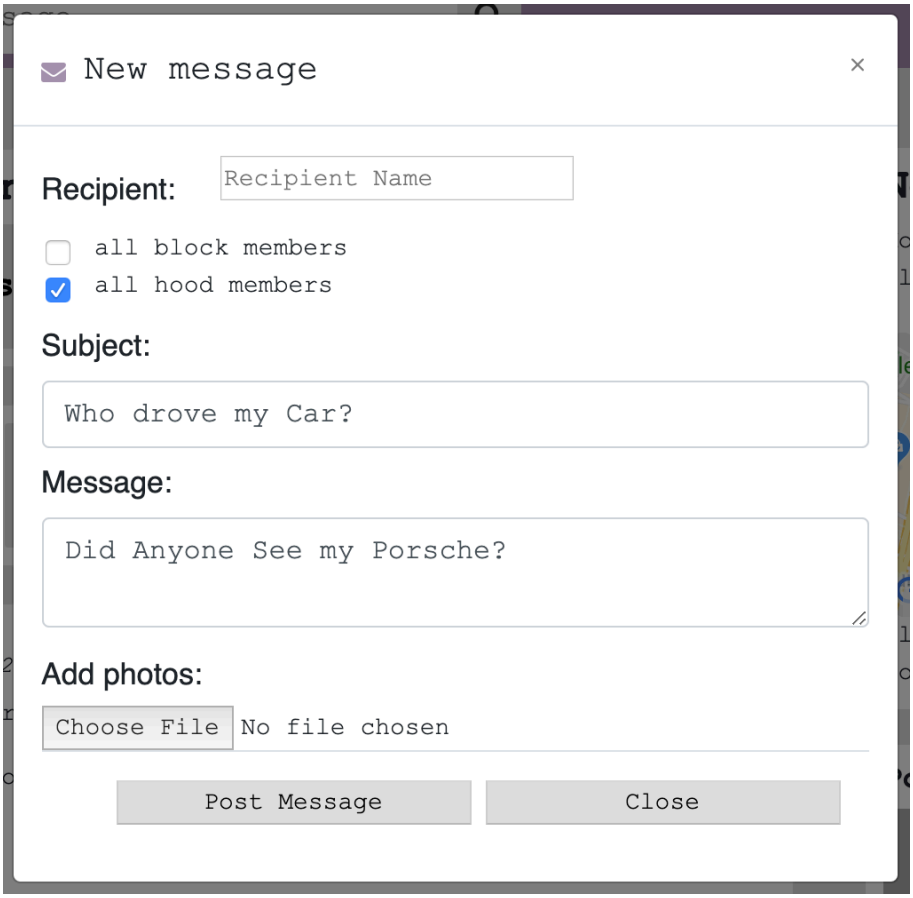


(4) New User Home page (regular status):

The screen below is the home page for regular user. The left column shows (1). A button that allow user the compose new messgaes. (2). Three buttons that let users to select the threads they want to see. (3). All the messages in the selected thread. The right column shows Neighbor and Block info.



User can post new messages by clicking post message button. User can select recipient by typing their name. Or either send message to the entire block or entire neighbor.



In PHP we first compose message:

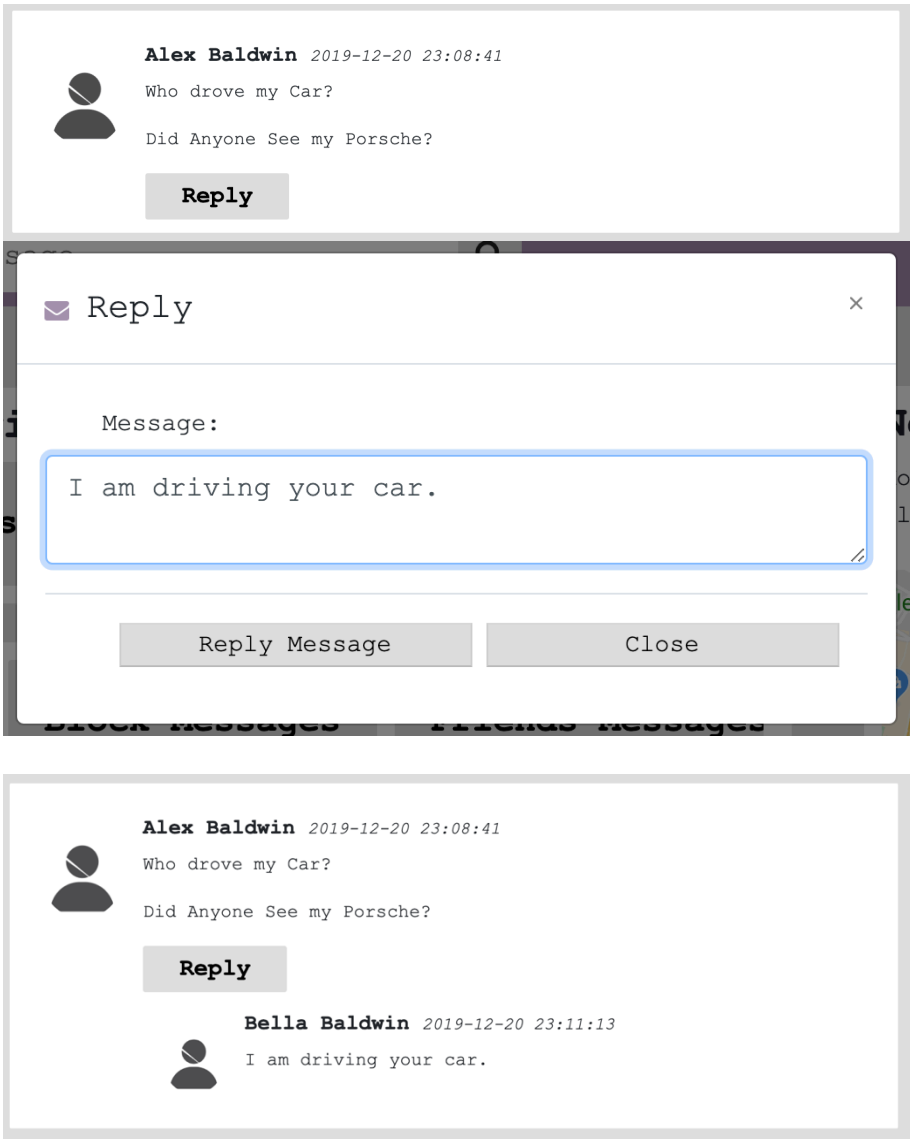
```php
function post_message_to_block() {
    $servername = "127.0.0.1";
    $username = "root";
    $password = "root";
    $dbname = "neighborhood";
    $conn = new mysqli($servername, $username, $password, $dbname);
    $sql = "CALL COMPOSE_MESSAGE('{$_POST['subject-name']}','{$_POST['message-text']}', {$_SESSION['current_uid']});";
    $compose = $conn->query($sql);
```

The direct message to target users.

```php
    $uid_block = $conn->query($sql);
    while ($row = $uid_block->fetch_assoc()) {

            $sql = "CALL DIRECT_MESSAGE({$_SESSION['current_uid']}, {$row[uid]});";
            $direct = $conn->query($sql);
    }
}
```

Other user in this Hood can view or reply to this message. New Reply will be shown in home page.



In PHP:

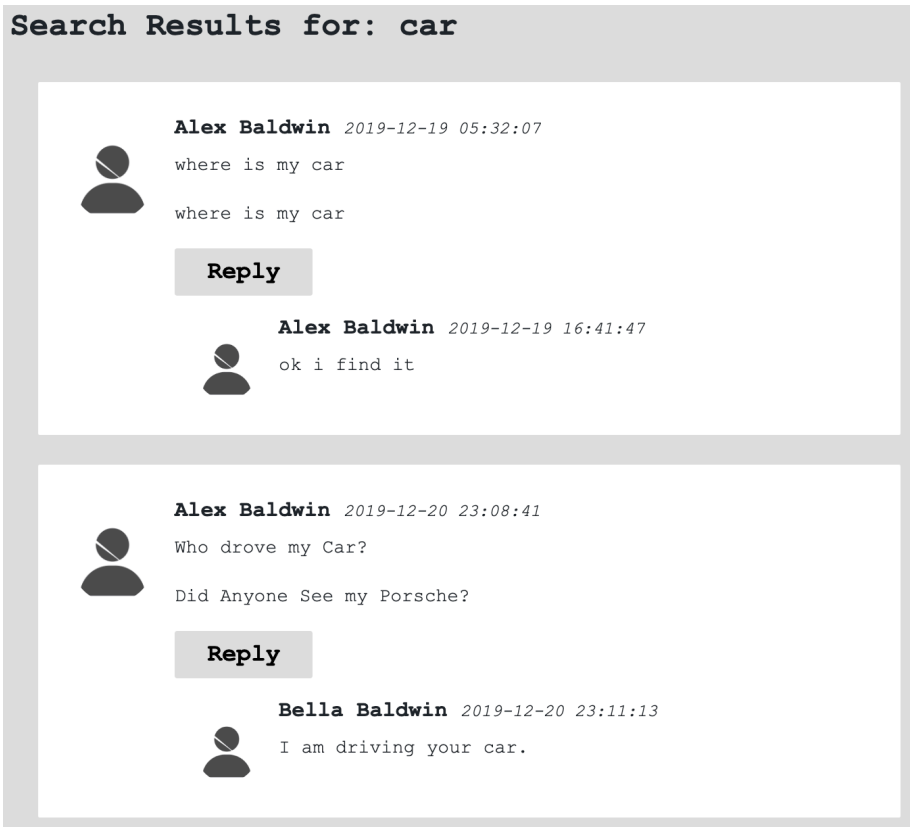
```php
function make_reply() {
    $servername = "127.0.0.1";
    $username = "root";
    $password = "root";
    $dbname = "neighborhood";
    $conn = new mysqli($servername, $username, $password, $dbname);

    $sql = "CALL COMPOSE_MESSAGE('reply','{$_POST['reply-text']}', {$_SESSION['current_uid']});";
    $compose = $conn->query($sql);

    $sql = "CALL REPLY_MESSAGE({$_SESSION['current_uid']}, {$_POST["reply_to_mid"]});";
    $direct = $conn->query($sql);
}
```

(5) Select Thread (Search by keyword):

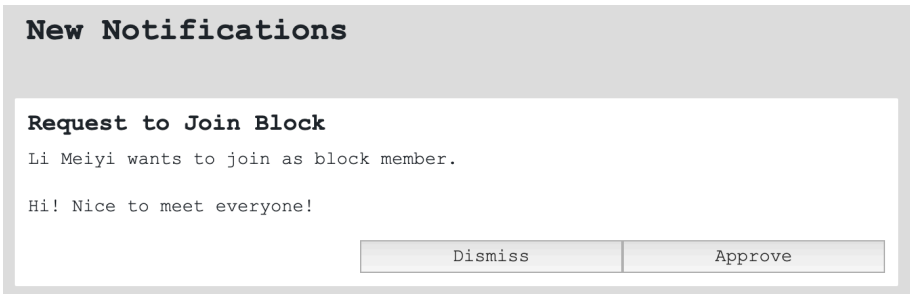
In the screen below, all message containing "Car" will be shown in main page.



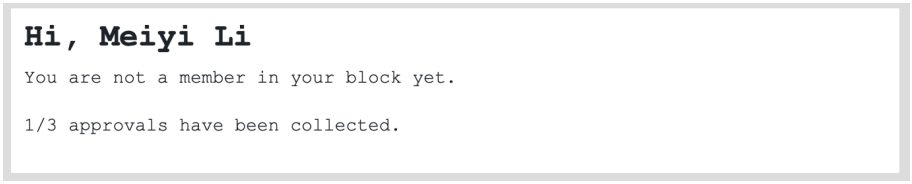
(6)Notification:

The bell icon indicates notification and number on right corner means the number of unread notifications.

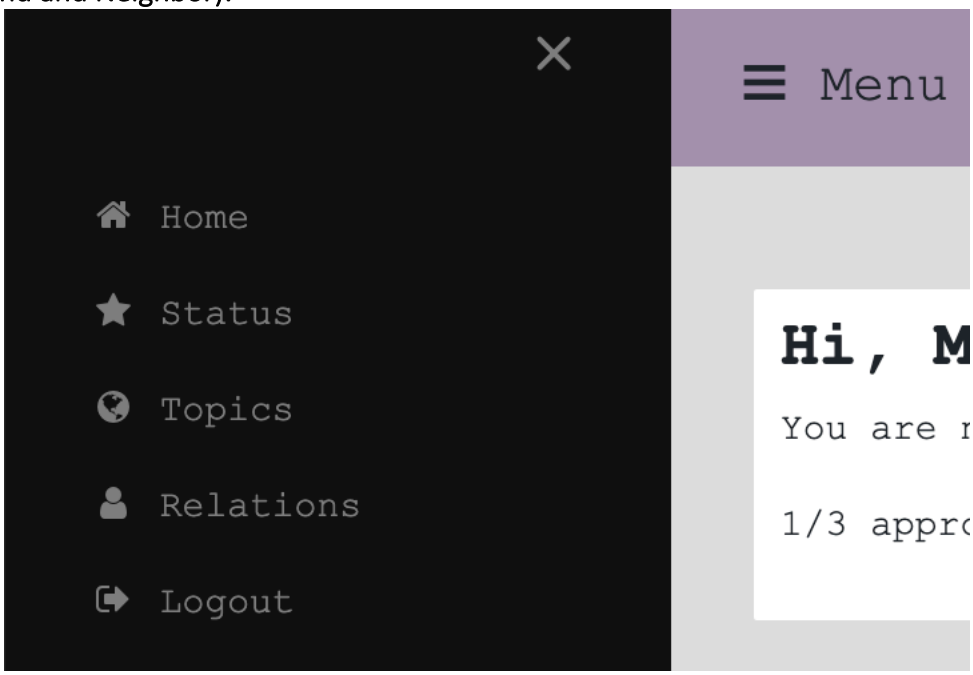
Notification can be Join block request or friend request.



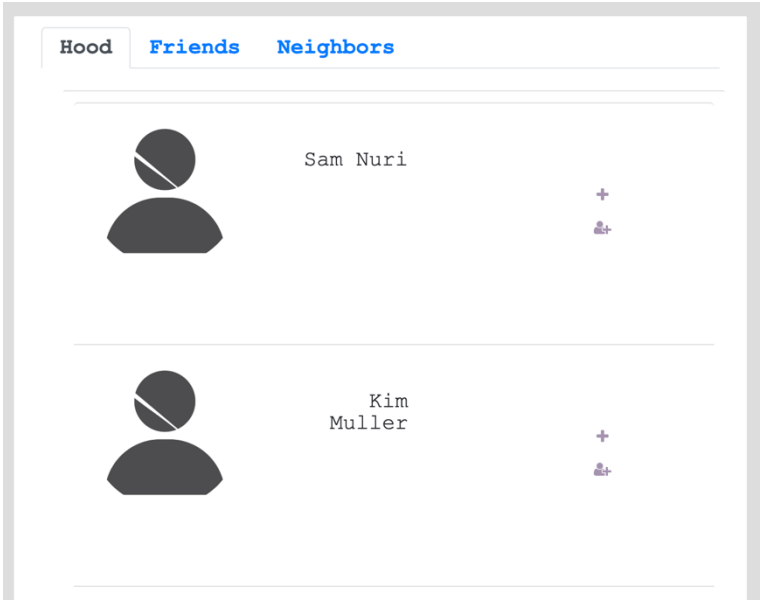
On approval, the request number on the sender's page will update.



(6)Relation (Friend and Neighbor):



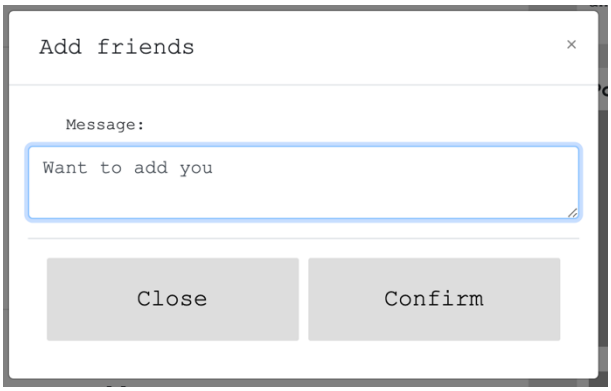
All member in Hood:

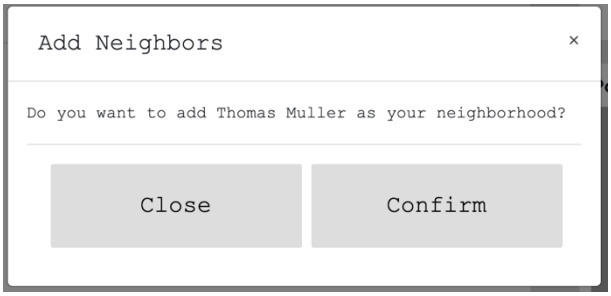


Friends:

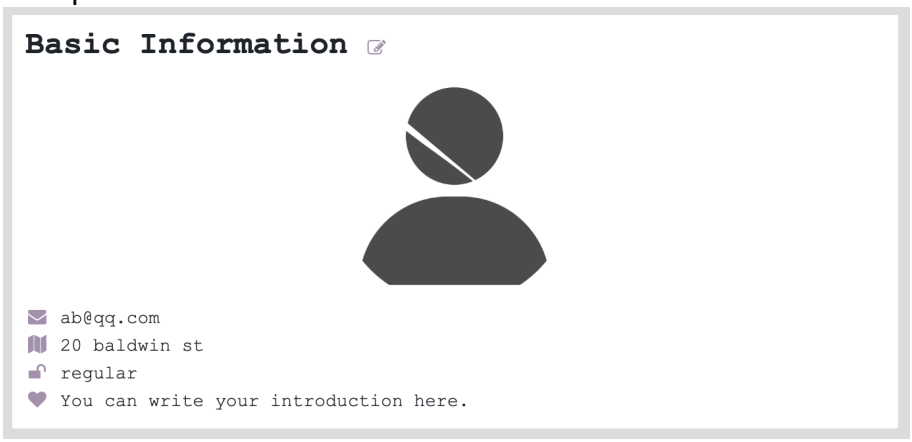
Neighbor

User can also add friend or neighbor for this list by click on the plus button.

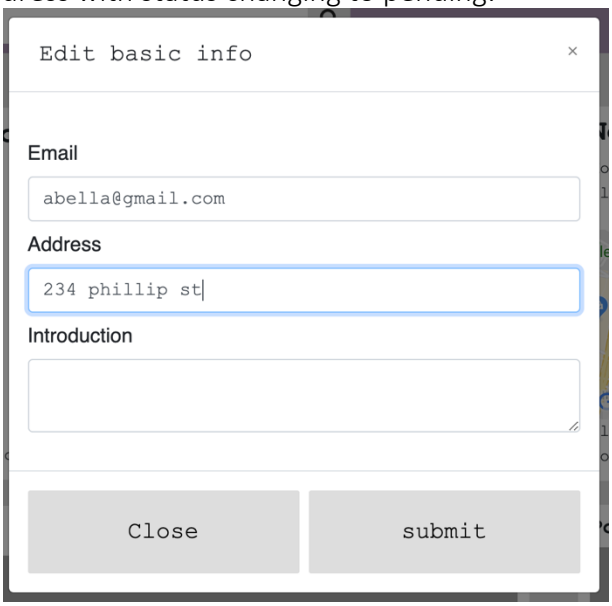




(7) Regular user update profile



Update email and Update address with status changing to pending:



Update image: