



School *of* Computing

CS2102

Project Team 28

Topic C: Stuff Sharing

Choong Yong Xin (A0171596U)

Lim Yi Hui Donna (A0172826Y)

Low Lin Hui Louisa (A0173545B)

Shi Kai Ning (A0172053N)

# Table of Contents

<b>1. Introduction</b>	<b>2</b>
1.1 Objective	2
1.2 Team Responsibilities	2
<b>2. Functionalities</b>	<b>3</b>
2.1 Login Page	3
2.2 Administrator Functions	4
2.3 User Functions	5
<b>3. ER Model</b>	<b>10</b>
<b>5. Triggers</b>	<b>15</b>
5.1 Trigger 1: insertBid	15
5.2 Trigger 2: placesBid	15
5.3 Trigger 3: changePassword	16
<b>6. Interesting SQL Queries</b>	<b>18</b>
<b>7. Software Tools used</b>	<b>19</b>
<b>8. Difficulties Faced and Lessons Learnt</b>	<b>19</b>
8.1 Difficulties Faced	19
8.2 Lessons Learnt	19

# 1. Introduction

## 1.1 Objective

Our application, ShareIt aims to provide an online stuff sharing service to facilitate convenient searching, borrowing and lending of items for our users. Users who wish to lend their items can create a post with the item for other users to bid or borrow for free. The user with the highest bid will be guaranteed a successful loan. Users who want to borrow items can also create requests so that other users are able to contact them if they happen to have the item. Users can also leave reviews for each other after loaning the items.

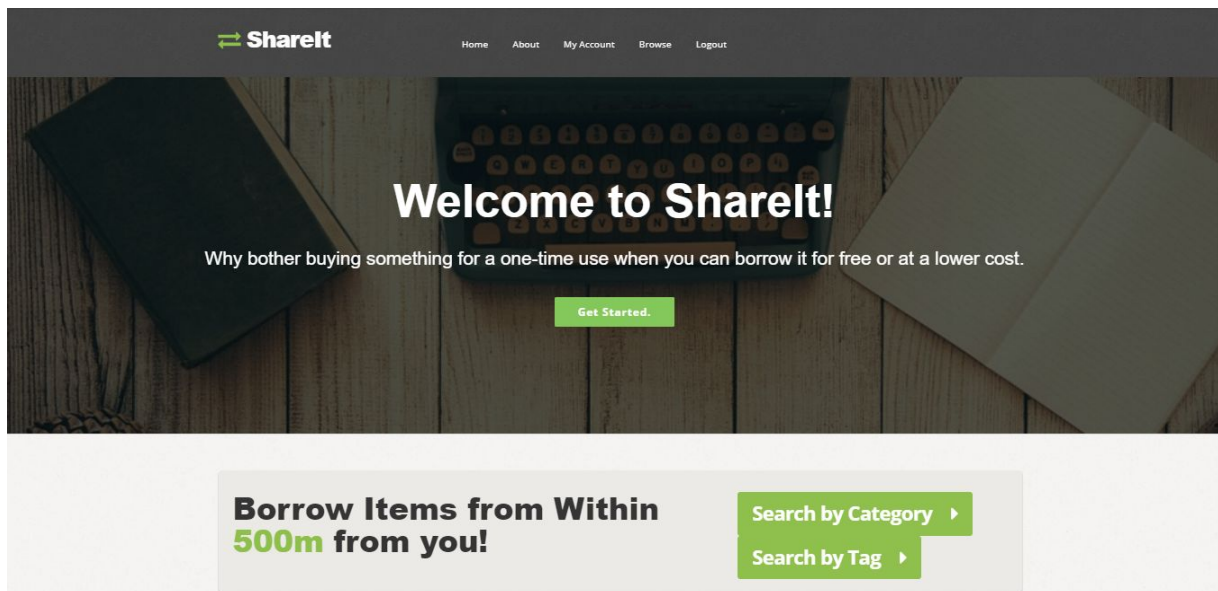


Figure 1: ShareIt Application Main Menu

## 1.2 Team Responsibilities

Team Member	Responsibilities
Louisa	Bidding, Handle Auction System, Create Triggers
Donna	Loans, Notifications, Profile, Register, Create ER Model
Yong Xin	Requests, Posts, Categories, Tags
Kai Ning	Design User interface, Create Triggers, Reviews, Tags, Login

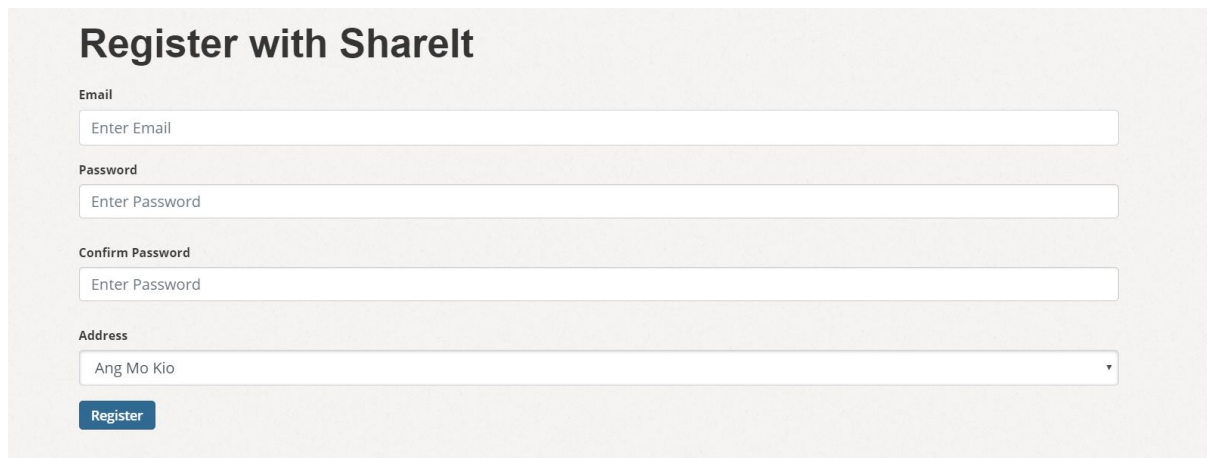
## 2. Functionalities

### 2.1 Login Page

#### Create an account

Visitors can register for an account on Sharelt by simply typing their email, password and address.

```
'INSERT INTO users(username, password, address) VALUES' + "(" + username +  
"`,`" + password + "`," + address + "`)" RETURNING userid";
```



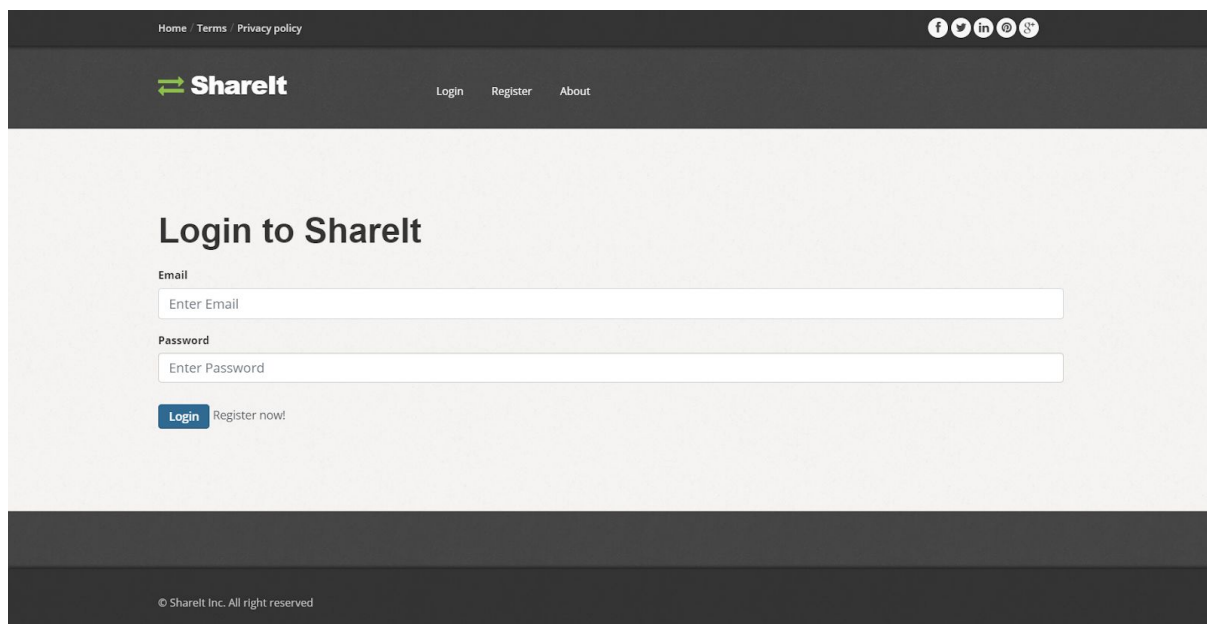
The registration form is titled "Register with Sharelt". It contains four input fields: "Email" with placeholder text "Enter Email", "Password" with placeholder text "Enter Password", "Confirm Password" with placeholder text "Enter Password", and "Address" with placeholder text "Ang Mo Kio". Below the address field is a blue "Register" button.

Figure 2: Registration Page

#### Login

Users can log into the system using their unique emails.

```
'SELECT userid, password, accountStatus FROM Users WHERE users.username =' + ""  
+ username + "";
```



The login form is titled "Login to Sharelt". It features a dark header with navigation links: "Home / Terms / Privacy policy", "Login", "Register", and "About". The main form has two input fields: "Email" with placeholder text "Enter Email" and "Password" with placeholder text "Enter Password". Below the password field is a blue "Login" button and a link "Register now!". The footer contains the text "© Sharelt Inc. All right reserved".

Figure 3: Login Page

## 2.2 Administrator Functions

### Create a New Category

Users with admin rights can create a new category by clicking on “Category Management” and then typing the category name and description for the new category.

```
'INSERT INTO categories(categoryName, description)
VALUES' + "(" + name + "'," + description + "');
```

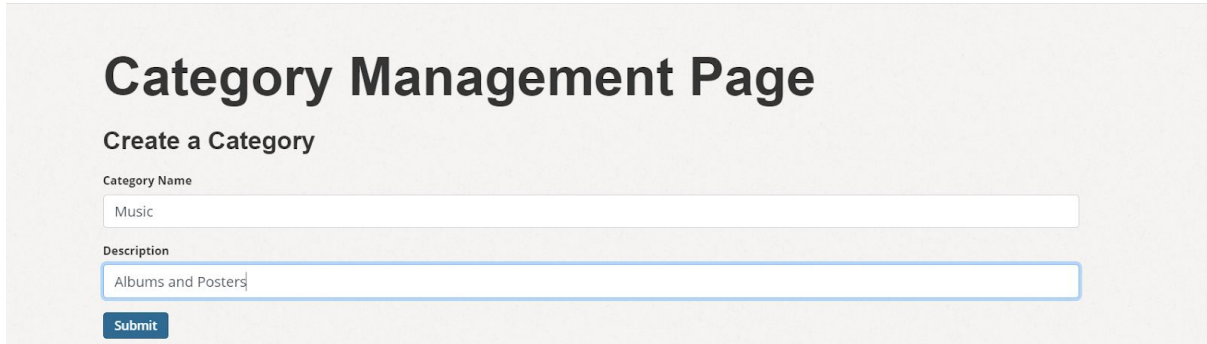
The screenshot shows a web page titled "Category Management Page". Below the title is a section "Create a Category". It contains two text input fields: "Category Name" with the value "Music" and "Description" with the value "Albums and Posters". A blue "Submit" button is located below the description field.

Figure 4: Category Management Page

### Delete an existing category

Admin can delete a category by typing the name of the category.

```
'DELETE FROM categories WHERE categoryname =' + "'" + deleteName + "'";
```

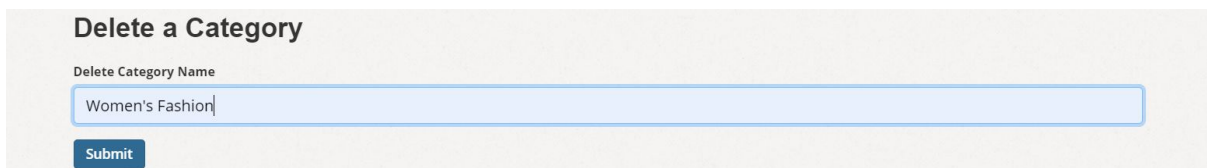
The screenshot shows a web page titled "Delete a Category". It contains a text input field labeled "Delete Category Name" with the value "Women's Fashion". A blue "Submit" button is located below the input field.

Figure 5: Delete Category Form

### View the list of Categories

Admin can view the existing list of categories.

```
'SELECT * FROM categories';
```

### Create a New Tag

Admin can create a new tag by clicking on “Tag Management” and then typing the tagname for the new tag.

```
'INSERT INTO tags(tagName) VALUES' + "(" + name + "');
```

### Delete an existing Tag

Admin can delete a category by typing the name of the category.

```
'DELETE FROM tags WHERE tagname =' + "'" + deleteName + "'";
```

### View the list of Tags

Admin can view the existing list of tags.

```
'SELECT * FROM tags';
```

## Display Auctions closing on current date

Admin can view the auctions set to close on the day itself.

```
"SELECT DISTINCT
users.username,auctions.highestbid,auctions.startingbid,posts.postid,
posts.productname,Categories.categoryname,posts.categoryid,
posts.description,posts.pickuppoint,posts.returnpoint,posts.startdate,
posts.enddate, auctions.auctionid, auctions.startingbid, auctions.highestbid,
auctions.endbiddate
FROM posts inner join Has ON posts.postid = Has.postid INNER JOIN Auctions ON
Has.auctionid = Auctions.auctionid INNER JOIN Categories ON
Categories.categoryid = Posts.categoryid LEFT OUTER JOIN Bids ON
auctions.highestbid = Bids.bidamount LEFT OUTER JOIN Places ON Places.bidid =
Bids.bidid LEFT OUTER JOIN Users ON Users.userid = Places.userid
WHERE auctions.endbiddate = current_date AND auctions.auctionstatus = '' +
"Open" + '' ORDER BY Posts.postid ASC";
```

## Close an Auction

Admin can view the close an auction set for closing on the day itself.

```
'UPDATE Auctions set ' + 'auctionStatus = ' + "'Close'" + ' WHERE auctionid =' +
"" + auctionId + "";
```

## 2.3 User Functions

### Create a Post and an Auction for the associated Post

To loan out a product, a user can create a post by clicking on “Go!” in the “Lend” box at the home page which brings the user to the “Create a Post” form.

```
'INSERT INTO posts(productName, categoryid, description, pickUpPoint,
returnPoint, startDate, endDate)
VALUES' + "(" + name + "','" + categoryid + "','" + description + "','" +
pickuppoint + "','" + returnpoint + "','" + startdate + "','" + enddate + "')
RETURNING postid";
```

If the loaner requires that the item be loaned with a price, the user can create an auction in the same page by clicking “Create Auction” before submitting the post.

```
'INSERT INTO auctions(startingBid, auctionstatus, endbiddate)
VALUES' + "(" + startBid + "','" + 'Open' + "','" + endBidding + "') RETURNING
auctionid";
```

The user does not need to create an auction for items that are free to loan.

### Delete a Post created by user

A user can delete a post by entering the postid in the “My Posts” page.

```
"DELETE FROM publishes WHERE postid = ' " + deleteId + 'and userid =' + userid;
```

## View all Posts

A user can view all posts by clicking on “Browse” or “Go!” in the “Posts” box of the main page.

```
"SELECT posts.postid, posts.productname, Categories.categoryname,
posts.categoryid, posts.description,posts.pickuppoint,
posts.returnpoint,posts.startdate,posts.enddate, auctions.auctionid,
auctions.startingbid, auctions.highestbid, auctions.endbiddate,
auctions.auctionStatus FROM posts left outer join Has on posts.postid =
Has.postid left join Auctions on Has.auctionid = Auctions.auctionid inner join
```

```
Categories on Categories.categoryid = Posts.categoryid WHERE
(auctions.auctionstatus = 'Open' or auctions.auctionstatus IS NULL) and
Posts.productname LIKE '%" + product + "%'" + " order by Posts.postid asc";
```

## View my Posts

A user can view posts created by himself/herself at “My Posts” under “My Account”.

```
"SELECT * FROM posts P natural join publishes U inner join Categories on
Categories.categoryid = P.categoryid where U.userid = '" + userid + "'";
```

## View Posts by Category

A user can view posts by category by clicking “Search by Category” on the main menu.

```
"SELECT Posts.postid, Posts.productname, Posts.description, Posts.pickuppoint,
Posts.returnpoint, Posts.startdate, Posts.enddate, auctions.auctionid,
auctions.highestbid, auctions.startingbid, auctions.endbiddate,
auctions.auctionStatus FROM posts natural join Belongs left outer join Has on
posts.postid = Has.postid left join Auctions on Has.auctionid =
Auctions.auctionid inner join Categories on Posts.categoryid =
Categories.categoryid where Belongs.categoryid = 1 AND (auctions.auctionstatus =
'Open' or auctions.auctionstatus IS NULL)"
/* 1 can be replaced by the categoryid for each category */
```

## View Posts by Tag

Users can view posts by tags by clicking on “Search by Tags” from the main menu.

Tags include:

### 1. Newest

```
"SELECT Posts.postid, Posts.productname, Posts.description,
Posts.pickuppoint, Posts.returnpoint, Posts.startdate, Posts.enddate from
Posts left outer join Has on Posts.postid = has.postid left outer join
Auctions on has.auctionid = Auctions.auctionid WHERE
auctions.auctionstatus = 'Open' or auctions.auctionstatus IS NULL order by
Posts.postid desc limit 10";
```

### 2. Near You

```
"SELECT * from Posts natural join Has natural join Auctions, Users where
(auctions.auctionstatus = 'Open' or auctions.auctionstatus IS NULL) AND
userid = " + "'" + userid + "'" + "and (pickuppoint = address or returnpoint
= address) AND (auctions.auctionstatus = 'Open' or auctions.auctionstatus
IS NULL)";
```

### 3. Popular

```
"with numbidders as (SELECT postid, count(*) as num from posts natural
join has natural join biddingfor group by postid) SELECT * from Posts
natural join Has natural join Auctions natural join numbidders WHERE
auctions.auctionstatus = 'Open' or auctions.auctionstatus IS NULL order by
num desc limit 10 ";
```

## Search for a Post by name

A user can search for a post by name in any of the pages displaying the posts.

```
"SELECT posts.postid, posts.productname, Categories.categoryname,
```

```
posts.categoryid, posts.description,posts.pickuppoint,
posts.returnpoint,posts.startdate,posts.enddate, auctions.auctionid,
auctions.startingbid, auctions.highestbid, auctions.endbiddate,
auctions.auctionStatus FROM posts left outer join Has on posts.postid =
Has.postid left join Auctions on Has.auctionid = Auctions.auctionid inner join
Categories on Categories.categoryid = Posts.categoryid WHERE
(auctions.auctionstatus = 'Open' or auctions.auctionstatus IS NULL) and
Posts.productname LIKE '%" + product + "%' " order by Posts.postid asc"
```

## View all Free Loans

A user can view all free loans by clicking on “Proceed to Free Loans” in the “Show Auction” button of the post.

```
'SELECT posts.postid, posts.productname, Categories.categoryname,
posts.categoryid, posts.description,posts.pickuppoint,
posts.returnpoint,posts.startdate,posts.enddate, auctions.auctionid,
auctions.startingbid, auctions.highestbid, auctions.endbiddate,
auctions.auctionStatus FROM posts left outer join Has on posts.postid =
Has.postid left join Auctions on Has.auctionid = Auctions.auctionid inner join
Categories on Categories.categoryid = Posts.categoryid where not exists (select
1 from Has where postid = posts.postid) order by Posts.postid asc'
```

## Create a Request

A user can create a request by clicking on “Go” in the “Requests” box and then clicking on “Create a Request”.

```
'INSERT INTO requests(productName, description, startdate, enddate,
contactnumber) VALUES' + "(" + name + "','" + description + "','" + startdate +
"', '" + enddate + "','" + contactnumber + "')" RETURNING requestid";
```

## Delete a Request created by user

A user can delete a request created by himself/herself on the “My Requests” page by inserting the requestid.

```
'DELETE FROM makes WHERE requestid =' + deleteId + 'and userid =' + userid;

'DELETE FROM requests WHERE requestid =' + deleteId;
```

## View all Requests

A user can view all requests by clicking on “Go” in the “Requests” box.

```
'SELECT * from requests natural join makes natural join users';
```

## View my Requests

A user can view requests created by himself/herself in the “My Requests” page.

```
"SELECT * FROM requests R, makes M WHERE R.requestid = M.requestid and userid =
'" + userid + "'";
```

## Place a Bid for a Post

A user can place a bid for the posts he/she wants to loan by clicking on the “Show Auction” button of the post. The user can enter the bid he/she wishes to place in the box as shown in Figure 2.



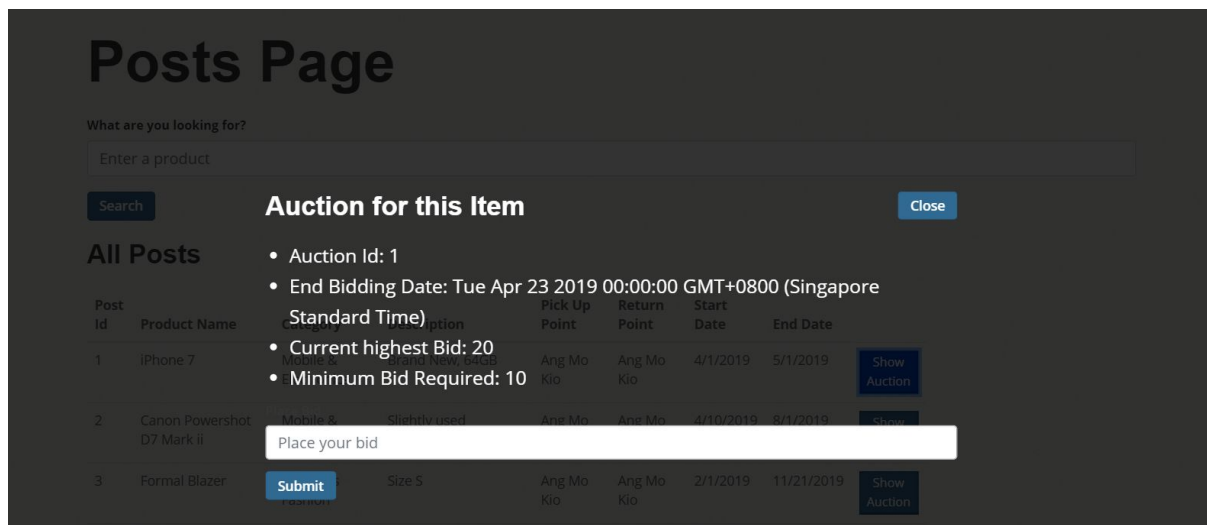


Figure 6: Bid Placing

The current highest bid will be updated as well.

```

"INSERT INTO bids(bidAmount) VALUES ('" + bidAmount + "') RETURNING bidid";

"INSERT INTO biddingfor(auctionid, bidid) VALUES ('" + auctionid + "','" + bidId
+ "')";

`UPDATE Auctions SET highestBid = ' + "'" + bidAmount + "'" + 'WHERE
auctions.auctionid = ' + auctionid + "`";

`INSERT INTO PLACES VALUES ' + "(" + "'" + userid + "','" + bidId + "','" + now +
"')";

```

## View my Bids

A user can view bids placed by himself/herself in the “My Bids” page.

```

"SELECT * FROM places NATURAL JOIN bids NATURAL JOIN biddingfor NATURAL JOIN
auctions NATURAL JOIN has where userid = '" + userid + "'";

```

## View my Loans

A user can view loans successfully acquired by himself/herself in the “My Loans” page.

```

"SELECT * FROM enablesLoan NATURAL JOIN secures where userid = '" + userid +
"'"';

```

## Receive Notifications

A user will receive notifications when he/she has successfully acquired for a loan. The user can view the notifications under the “Notifications” page.

```

"SELECT * FROM receivesnotifications where userid = '" + userid + "'";

```

## Search Users

A user can search for another user by clicking on “Search Users”.

```

`SELECT * FROM Users WHERE users.username LIKE ' + "%"+ username + "%' + 'AND
userid <> 1 AND userid <> ' + userid;
/*userid = 1 is the Admin*/

```

## Create a Review

A user can create a review for another user by clicking on “Review” and entering the userid of the user to be reviewed as well as the content and rating.

```
'INSERT INTO reviews(content, rating) VALUES' + "(" + reviewContent + "," + reviewRating + ") RETURNING reviewid";
```

```
'INSERT INTO Creates VALUES' + "(" + revieweeid + "," + reviewid + "," + userid + ")";
```

## Delete a Review created by user

A user can delete a review created by himself/herself on “My Reviews” page by inserting the reviewid.

```
'DELETE FROM reviews WHERE reviewid =' + "'" + deleteId + "';"  
'DELETE FROM creates WHERE reviewid =' + "'" + deleteId + "';"  
/* deleteId is the reviewid of the review to be deleted */
```

## View my Reviews of others

A user can view reviews created by himself/herself in the “My Reviews” page.

```
"SELECT * FROM reviews R, creates C inner join users U on C.reviewerid =  
U.userid WHERE R.reviewid = C.reviewid and revieweeid = '" + userid + '"
```

## View Reviews of Me

A user can view reviews of himself/herself created by other users in “Reviews of Me” Page.

```
"SELECT * FROM reviews R, creates C inner join users U on C.reviewerid =  
U.userid WHERE R.reviewid = C.reviewid and revieweeid = '" + userid +  
"'"
```

## Change address

A user can update their address by selecting a MRT station from the drop-down list in “Profile” page.

```
'UPDATE users set ' + 'address = ' + "'" + newAddress + "'" + ' WHERE userid =' +  
+ "'" + userid + "'";
```

## Change password

A user can change their password by entering their original and new password in ‘Profile’ page.

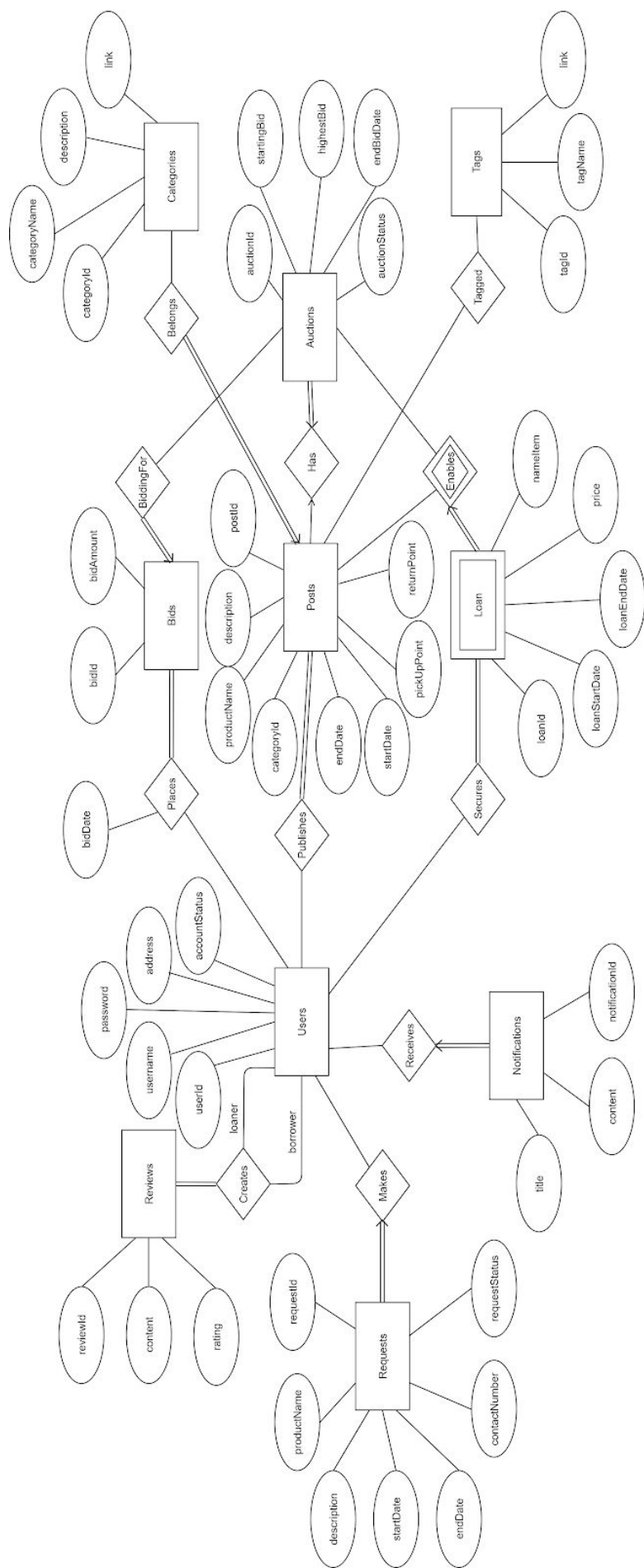
```
'UPDATE users set ' + 'password = ' + "'" + newPassword + "'" + ' WHERE userid ='  
= ' + "'" + userid + "'"
```

## Delete account

A user can close their account by entering their password in “Profile” page.

```
"UPDATE users SET accountstatus = 'Close' WHERE userid =" + "'" + userid + "'"
```

### 3. ER Model



## 4. Relational Schema

```
CREATE TABLE Tags (  
    tagid SERIAL,  
    tagName VARCHAR(30) UNIQUE,  
    link VARCHAR(30),  
    PRIMARY KEY (tagid)  
);  
  
CREATE TABLE Bids (  
    bidid SERIAL,  
    bidAmount REAL,  
    PRIMARY KEY (bidid)  
);  
  
CREATE TYPE auctionENUM as ENUM('Open','Close');  
  
CREATE TABLE Auctions (  
    auctionId SERIAL,  
    startingBid DECIMAL,  
    highestBid DECIMAL,  
    auctionStatus auctionENUM,  
    endBidDate TIMESTAMP,  
    PRIMARY KEY (auctionId)  
);  
  
CREATE TYPE statusENUM as ENUM('Accepted','Pending', 'Rejected', 'Withdrawn');  
  
CREATE TABLE Categories (  
    categoryid SERIAL,  
    categoryName VARCHAR(30) UNIQUE,  
    description VARCHAR(100),  
    link VARCHAR(30),  
    PRIMARY KEY (categoryid)  
);  
  
CREATE TABLE Posts (  
    postid SERIAL,  
    productName VARCHAR(30),  
    categoryid INTEGER,  
    description VARCHAR(30),  
    pickupPoint VARCHAR(30),  
    returnPoint VARCHAR(30),  
    startDate DATE,  
    endDate DATE,  
    PRIMARY KEY (postId),  
    FOREIGN KEY (categoryid) REFERENCES Categories  
);  
  
CREATE TABLE Requests (  
    requestid SERIAL,  
    productName VARCHAR(30),  
    description VARCHAR(30),  
    startDate DATE,  
    endDate DATE,  
    contactNumber VARCHAR(30),
```

```

    requestStatus    statusENUM,
    PRIMARY KEY (requestid)
);

CREATE TYPE ratingENUM as ENUM('Positive', 'Neutral', 'Negative');

CREATE TABLE Reviews (
    reviewid SERIAL,
    content VARCHAR(300),
    rating ratingENUM,
    PRIMARY KEY (reviewid)
);

CREATE TYPE userENUM as ENUM('Open','Close');

CREATE TABLE Users (
    userid SERIAL,
    username VARCHAR(40) UNIQUE,
    password VARCHAR(20),
    address VARCHAR(100),
    accountStatus userENUM DEFAULT 'Open',
    PRIMARY KEY (userid)
);

CREATE TABLE EnablesLoan (
    loanId SERIAL,
    nameItem VARCHAR(30),
    loanStartDate DATE,
    loanEndDate DATE,
    price REAL,
    auctionId SERIAL UNIQUE,
    postId SERIAL,
    PRIMARY KEY (loanId),
    FOREIGN KEY (auctionId) references AUCTIONS,
    FOREIGN key (postId) references POSTS
);

CREATE TABLE Places (
    userid SERIAL,
    bidid SERIAL,
    bidDate DATE,
    PRIMARY KEY (userid, bidid),
    FOREIGN KEY (userid) REFERENCES Users,
    FOREIGN KEY (bidid) REFERENCES Bids
);

CREATE TABLE BiddingFor (
    auctionid SERIAL,
    bidid SERIAL,
    PRIMARY KEY (auctionid, bidid),
    FOREIGN KEY (auctionid) REFERENCES Auctions,
    FOREIGN KEY (bidid) REFERENCES Bids
);

CREATE TABLE Belongs (
    postId SERIAL,

```

```

categoryid SERIAL,
primary key (postid, categoryid),
foreign key (categoryid) references Categories,
foreign key (postid) references Posts
);

CREATE TABLE Tagged (
    tagid SERIAL,
    postid SERIAL,
    primary key (tagid, postid),
    foreign key (tagid) references Tags,
    foreign key (postid) references Posts
);

CREATE TABLE Has (
    postId SERIAL,
    auctionId SERIAL,
    PRIMARY KEY (postId, auctionId),
    FOREIGN KEY (postId) references POSTS,
    FOREIGN KEY (auctionId) references AUCTIONS
);

CREATE TABLE Publishes (
    postid SERIAL,
    userid SERIAL,
    PRIMARY KEY (postid),
    FOREIGN KEY (postid) REFERENCES Posts,
    FOREIGN KEY (userid) REFERENCES Users
);

CREATE TABLE Makes (
    userid SERIAL,
    requestid SERIAL,
    PRIMARY KEY (userid, requestid),
    FOREIGN KEY (userid) REFERENCES Users,
    FOREIGN KEY (requestid) REFERENCES Requests
);

CREATE TABLE Creates (
    revieweeid SERIAL,
    reviewid SERIAL,
    reviewerid SERIAL,
    PRIMARY KEY (revieweeid, reviewid, reviewerid),
    FOREIGN KEY (revieweeid) REFERENCES Users(userid),
    FOREIGN KEY (reviewid) REFERENCES Reviews,
    FOREIGN KEY (reviewerid) REFERENCES Users(userid)
);

CREATE TABLE ReceivesNotifications (
    userid SERIAL,
    notificationid SERIAL,
    title VARCHAR(50),
    content VARCHAR(300),
    PRIMARY KEY (userid, notificationid),
    FOREIGN KEY (userid) REFERENCES Users
);

```

```
CREATE TABLE Secures (  
    loanId SERIAL,  
    userId SERIAL,  
    PRIMARY KEY (loanId,userId),  
    FOREIGN KEY (loanId) references ENABLESLOAN,  
    FOREIGN KEY (userId) references USERS  
);
```

## 5. Triggers

### **5.1 Trigger 1: insertBid**

The insertBid Trigger is triggered when actions involving insert and update on rows in the biddingFor Table. The insertBid trigger checks if the new bid being placed by the user for the auction it is bidding for is of a higher value than the current highest bid for that auction, if it is greater, it will be inserted else it will abort the insertion and delete the new bid that was inserted.

```
--Trigger 1
CREATE OR REPLACE FUNCTION checkHighest()
RETURNS TRIGGER AS
$$
DECLARE maximal DECIMAL;
        belongAuction REAL;
BEGIN
SELECT highestbid INTO maximal
FROM Auctions
WHERE NEW.auctionid = Auctions.auctionid;
SELECT bidamount INTO belongAuction
FROM Bids
WHERE Bids.bidid = NEW.bidid;
IF (belongAuction > maximal) OR (maximal is NULL)
THEN
RETURN NEW;
ELSE
DELETE FROM Bids where bidid = NEW.bidid;
RETURN NULL;
END IF ;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER insertBid
BEFORE INSERT OR UPDATE
ON BiddingFor
FOR EACH ROW
EXECUTE PROCEDURE checkHighest();
```

### **5.2 Trigger 2: placesBid**

The placesBid trigger is triggered when actions involving insert and update on rows in the places table. The placesBid trigger checks if the user has a current existing bid for the particular auction that the user is bidding for. If there is no previous existing bid for the auction by the user, the new bid is left alone. Else, the previous bid for that auction and its associating relations such as places and biddingfor is deleted, replaced by the new bid.

```
--Trigger 2
CREATE OR REPLACE FUNCTION checkDuplicate()
RETURNS TRIGGER AS
$$
DECLARE auctionSelected INTEGER;
```



```

        anotherBid INTEGER;
BEGIN
select Auctions.auctionid INTO auctionSelected
from places natural join Bids natural join biddingfor natural join Auctions
where places.bidid = NEW.bidid;
select Places.bidid INTO anotherBid
from places natural join Bids natural join biddingfor natural join Auctions
where Auctions.auctionid = auctionSelected and places.bidid <> NEW.bidid and
Places.userid = NEW.userid;
if (anotherBid is not null) THEN
DELETE FROM places where places.bidid = anotherBid;
DELETE FROM biddingfor where biddingfor.bidid = anotherBid;
DELETE FROM bids where bids.bidid = anotherBid;
RETURN NEW;
else
RETURN NEW;
END IF;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER placesBid
AFTER INSERT OR UPDATE
ON Places
FOR EACH ROW
EXECUTE PROCEDURE checkDuplicate();

```

### **5.3 Trigger 3: changePassword**

The changePassword trigger is triggered when there is an attempt to change a user's password.

The trigger checks that the new password that the user enters is not the same as the current password of the user, if it is the same, null is returned.

Else, the old password is replaced with the new password.

```

--Trigger 3
CREATE OR REPLACE FUNCTION def_password()
RETURNS TRIGGER AS
$$
DECLARE oldPassword VARCHAR;
        oldAddress VARCHAR;
        oldAccountStatus userENUM;
BEGIN
    SELECT Users.password INTO oldPassword
    FROM Users
    WHERE Users.userId = NEW.userid;
    SELECT Users.address INTO oldAddress
    FROM Users
    WHERE Users.userId = NEW.userid;
    SELECT Users.accountStatus INTO oldAccountStatus
    FROM Users
    WHERE Users.userId = NEW.userid;
    IF (NEW.password = oldPassword) AND (NEW.address = oldAddress) AND
(NEW.accountStatus = oldAccountStatus) THEN
        RETURN NULL;

```

```
        ELSE
            RETURN NEW;
        END IF;
    END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER changePassword
BEFORE UPDATE
ON Users
FOR EACH ROW
EXECUTE PROCEDURE def_password();
```

## 6. Interesting SQL Queries

### Display Popular posts: Those with highest number of non-distinct bidders

Reasoning for using non-distinct bidders: The higher the number of bidders re-bidding indicates the higher demand they have for the item, showing its popularity.

```
'with numbidders as (SELECT postid, count(*) as num from posts natural join has
natural join biddingfor group by postid) SELECT * from Posts natural join
numbidders order by num desc limit 10';
```

### Display Auctions closing on current date

```
"SELECT DISTINCT
users.username,auctions.highestbid,auctions.startingbid,posts.postid,
posts.productname,Categories.categoryname,posts.categoryid,
posts.description,posts.pickuppoint,posts.returnpoint,posts.startdate,
posts.enddate, auctions.auctionid, auctions.startingbid, auctions.highestbid,
auctions.endbiddate
FROM posts inner join Has ON posts.postid = Has.postid INNER JOIN Auctions ON
Has.auctionid = Auctions.auctionid INNER JOIN Categories ON
Categories.categoryid = Posts.categoryid LEFT OUTER JOIN Bids ON
auctions.highestbid = Bids.bidamount LEFT OUTER JOIN Places ON Places.bidid =
Bids.bidid LEFT OUTER JOIN Users ON Users.userid = Places.userid
WHERE auctions.endbiddate = current_date AND auctions.auctionstatus = '\" +
"Open" + "' ORDER BY Posts.postid ASC";
```

### Display Posts belonging to a certain category

```
"SELECT Posts.postid, Posts.productname, Posts.description, Posts.pickuppoint,
Posts.returnpoint, Posts.startdate, Posts.enddate,auctions.auctionid,
auctions.highestbid,auctions.startingbid, auctions.endbiddate,
auctions.auctionStatus FROM posts natural join Belongs left outer join Has on
posts.postid = Has.postid left join Auctions on Has.auctionid =
Auctions.auctionid inner join Categories on Posts.categoryid =
Categories.categoryid where Belongs.categoryid = 4 AND (auctions.auctionstatus =
'Open' or auctions.auctionstatus IS NULL)"
/* 1 can be replaced by the categoryid for each category */
```

## 7. Software Tools used

- Platform: nodeJs
- Framework: express
- Template Engine: ejs
- CSS Framework: bootstrap
- CSS Preprocessor: css
- JavaScript Framework: nodeJs
- Build Tool: none
- Unit Testing: none
- Database: postgresSQL
- Authentication: none
- Deployment: none

## 8. Difficulties Faced and Lessons Learnt

### **8.1 Difficulties Faced**

- Node.js crashes often when we are testing out the different functionalities as the order of the different methods executed affects the loading of the pages.
- Hard to implement external npm such as the login plugins as instructions given on those sites are hard to understand.
- It is difficult to also implement some of the more complex functions in node.js as we are not very proficient in using it and time does not permit us to explore, resulting in implementations of one function to cause others to break down.

### **8.2 Lessons Learnt**

#### **New Framework and Technology**

Picking up nodeJs, express, ejs and bootstrap from scratch in a short time posed a huge challenge for most of us as we did not have prior experience in using these technologies. From this project, we learnt how to be an independent learner by searching online for resources to learn how to use these software tools ourselves. Such skills are essential in the IT industry as technology is always evolving and as future IT professionals, we need to learn to keep learning and improve ourselves to keep up.

#### **Schema Design and SQL Query**

In order to ensure that the associations between different entities were accurate, the schema was repeatedly revised to uphold better data integrity and this was rather tedious. Coming up with complex SQL queries was difficult as all the team members are new to PostgreSQL. This was especially so for queries that required data from different tables.

The project provides us with opportunities to actually explore different ways to implement triggers to suit the needs of our project, it was a challenge to overcome as there are many relations and associations involved. However, when the trigger is successfully implemented, it is a lesson well-learned.