CS2102 Design Report
AY2018/2019 Semester 1

Topic C: Stuff Sharing

Group 68
Members:
Jia Yilin (A0143648Y)
Li Shuli (A0148013R)
Tey Xin Hui (A0159783L)
Tian Xin (A0148036H)
Zhao Tianze (A0147989B)

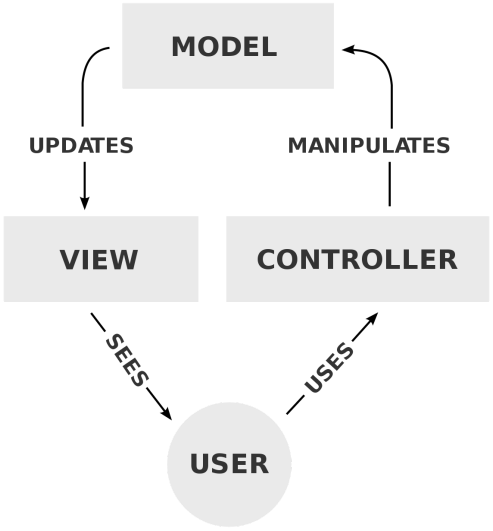
1 Introduction

In this project, our target is to build a stuff sharing website with connect to a SQL database. The system allows people to borrow or lend stuff that they own (tools, appliances, furniture or books) either free or for a fee. Users advertise stuff available (what stuff, whether delivery is offered, when it is available, etc.) or can browse the available stuff and bid to borrow some stuff. The stuff owner or the system (your choice) chooses the successful bid. Each user has an account and they have access to manage their own entries that they own, such as posts they created. Administrators can create, modify and delete all entries.

1.1 Implementation Specification

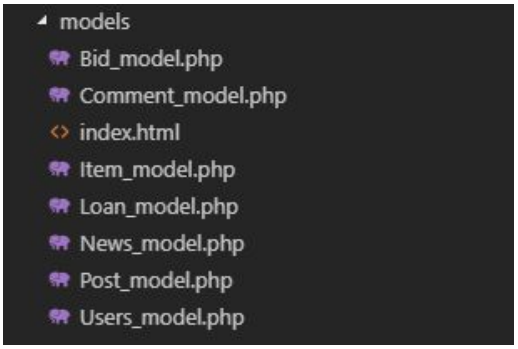
In our project, we develop both front end and back end by PHP and use Javascript to help front end. We choose PostgreSQL for database performance and choose Codeigniter MVC PHP framework(shown below) as it is one of the best one for small size products.

MVC Framework:

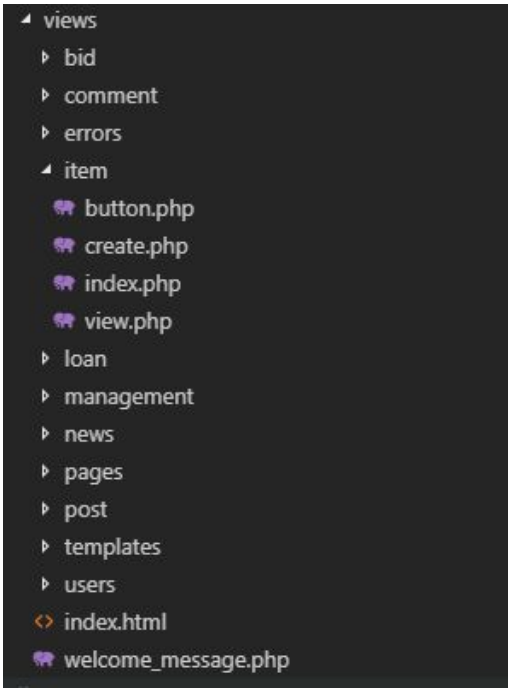


In particular, inside the framework, model part takes charge of handling all the communication with database. Therefore, for each entity table, we have a corresponding file inside this model folder.

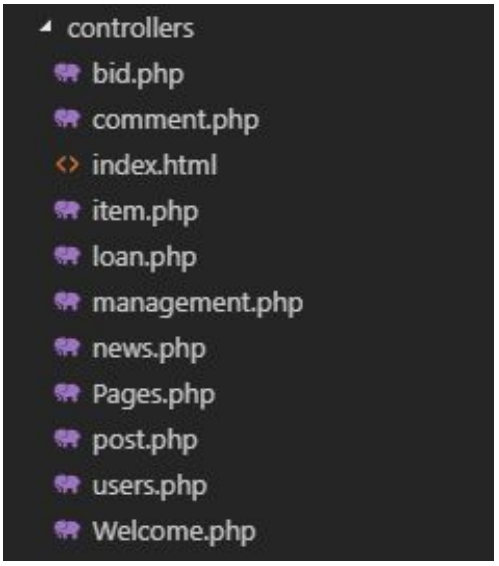
Model:



View:



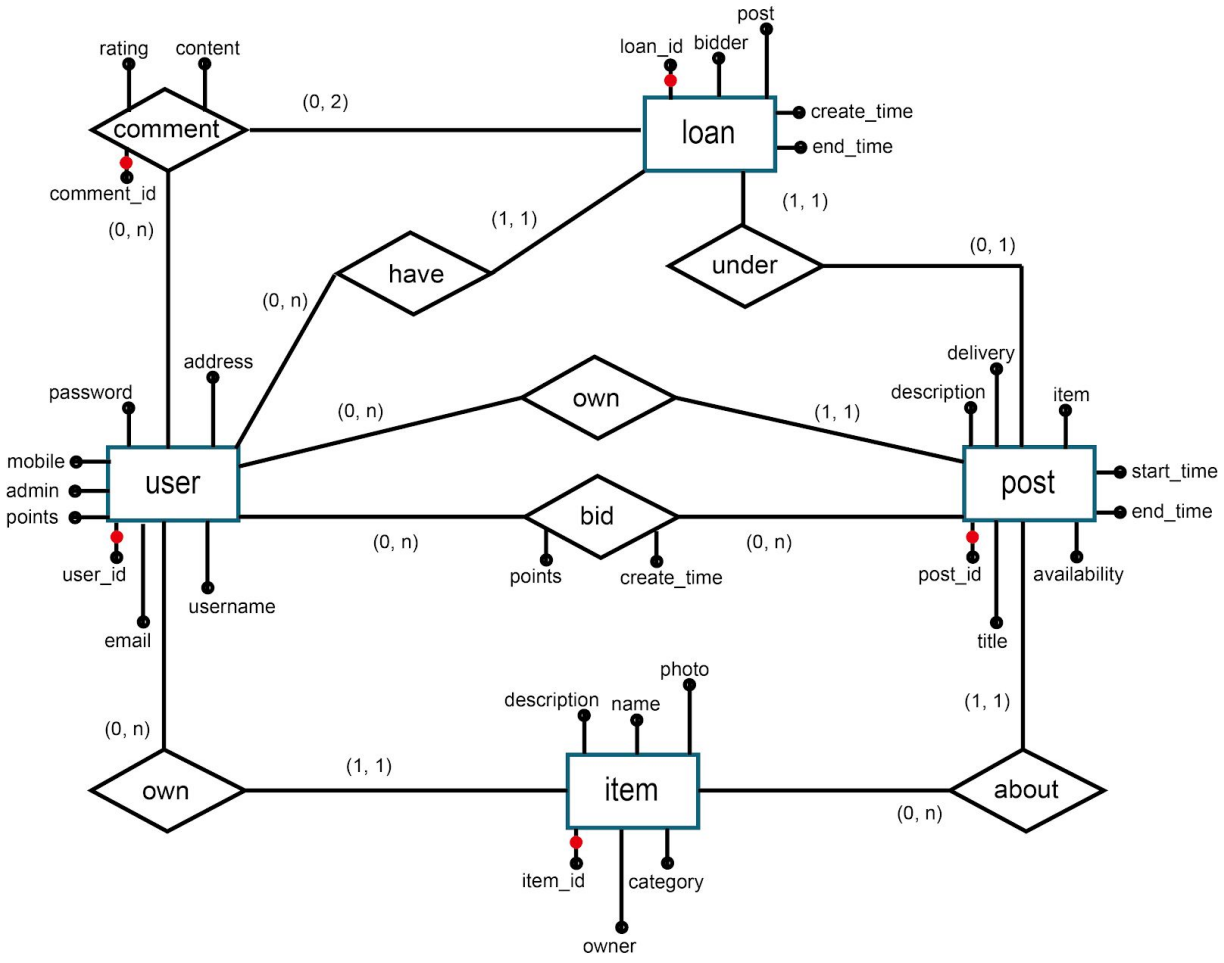
Controller:



Since in our database design, we have two types of users, admin and non-admin. Admin users have access to the Switch-User function to manage all entries.

2 Database Design

2.1 Entity-Relationship Diagram



2.2 Entities

a. Users Table

Attributes	Domain
user_id	SERIAL
username	VARCHAR(64)
mobile	VARCHAR(64)
email	VARCHAR(64)
password	VARCHAR(200)
admin	BOOLEAN

points	NUMERIC
address	VARCHAR(64)

b. Item Table

Attributes	Domain
item_id	SERIAL
name	VARCHAR(64)
owner	INTEGER
category	VARCHAR(64)
description	VARCHAR(200)
photo	BYTEA

c. Post Table

Attributes	Domain
post_id	SERIAL
title	VARCHAR(64)
item	INTEGER
start_time	TIMESTAMP
end_time	TIMESTAMP
description	VARCHAR(200)
delivery	BOOLEAN
availability	BOOLEAN

d. Bid Table

Attributes	Domain
bidder	INTEGER
post	INTEGER
points	NUMERIC
create_time	TIMESTAMP

e. Loan Table

Attributes	Domain
loan_id	SERIAL
bidder	INTEGER
post	INTEGER
start_time	TIMESTAMP
end_time	TIMESTAMP

f. Comment Table

Attributes	Domain
comment_id	SERIAL
loan	INTEGER
user_name	INTEGER
content	VARCHAR(200)
rating	NUMERIC

2.3 Rational Schema

```
CREATE TABLE users(
user_id SERIAL PRIMARY KEY ,
username VARCHAR(64) NOT NULL,
mobile VARCHAR(64) NOT NULL,
email VARCHAR(64) UNIQUE,
password VARCHAR(64) NOT NULL,
admin BOOLEAN NOT NULL,
points NUMERIC DEFAULT 20 CHECK (points >= 0),
address VARCHAR(64));


CREATE TABLE item(
item_id SERIAL PRIMARY KEY ,
name VARCHAR(64) NOT NULL,
owner INTEGER REFERENCES users(user_id) ON DELETE CASCADE,
category VARCHAR(64) NOT NULL DEFAULT 'ALL',
description VARCHAR(64),
photo bytea);


CREATE TABLE post(
post_id SERIAL PRIMARY KEY,
title VARCHAR(64) NOT NULL,
item INTEGER NOT NULL REFERENCES item(item_id) ON DELETE CASCADE,
start_time TIMESTAMP DEFAULT NOW(),
end_time TIMESTAMP,
description VARCHAR(200),
delivery BOOLEAN NOT NULL,
availability BOOLEAN);


CREATE TABLE bid(
bidder INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
```

```
post INTEGER NOT NULL REFERENCES post(post_id) ON DELETE CASCADE,
points NUMERIC CHECK (points >= 0),
create_time TIMESTAMP DEFAULT NOW(),
PRIMARY KEY(bidder, post));

CREATE TABLE loan(
loan_id SERIAL PRIMARY KEY,
bidder INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
post INTEGER NOT NULL REFERENCES post(post_id) ON DELETE CASCADE,
start_time TIMESTAMP DEFAULT Now(),
end_time TIMESTAMP NOT NULL);


CREATE TABLE comment(
comment_id SERIAL PRIMARY KEY,
loan INTEGER NOT NULL REFERENCES loan(loan_id) ON DELETE CASCADE,
user_name INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
content VARCHAR(200),
rating NUMERIC NOT NULL CHECK (rating>=0 AND rating<=5));
```

2.4 Schema Functions

The *users* table includes user's information such as *user_id, username, email* and *password* etc. And users with TRUE *admin* entity value are able to manage all the users, items, bids and posts etc. The normal users can only update their own items, bids, posts, loans and comments. To be more specific, as a lender, he/she can make a post about one of his/her items and update details. During the indicated bidding period, he/she has the right to choose his/her prefered bidder to borrow the item. Otherwise, the system will automatically choose the bid with highest points. As a borrower, he/she can browse all the available post and bid for the items.

The *item* table includes item's information such as *item_id*, *name* and its *owner* etc. The item availability can be obtained by checking through statuses of all the relevant posts. If there is only one available post, then the item is not available to post again. Otherwise, it's available for a lender to post.

The *post* table includes post's information such as post_id, its relevant *item*, *title, availability* and bidding period indicated by *start_time* and *end_time* etc. Borrowers can make bid according to the post information given.

The *bid* table includes *bidder*(reference from *user_id*), *post*(reference from *post_id*) , *bidding_points* and *created_time*. And it can be identified by bidder and post. Users are able to view and update their own bids and if they are interested in an item's post, they also can view all bids about this post.

The *loan* table includes loan information such as *loan_id, post* about the loaned item, user loaning this item and time of borrow and return. A loan will be created after a bidder is chosen in a post. If the item has not been returned, *end_time* in table will be None.

The *comment* table includes comment's information such as *comment_id*, *user_name*, *loan* and *rating* etc. Two users in a deal can make several comments with each other.

3. SQL Queries

3.1 Simple Queries

This simple query can find out all the items that belong to user with userid 68. This will be displayed when this user wants to view, add or delete his own items.

```
SELECT *
FROM item
WHERE owner = 68;
```

This simple query returns all the bidding history of post with id 68 including bidder, points they put and the time they bid this post. Result is ordered by bidding points from the highest to the lowest. This will be displayed when a user wants to select a successful bidder.

```
SELECT bidder, points, create_time
FROM bid
WHERE post = 68
ORDER BY points DESC;
```

This simple query returns all the available post_id belong to user with userid 68. This will be displayed when someone wants to view all the available posts owned by a particular user.

```
SELECT p.post_id
FROM post p, item i
WHERE p.item = i.item_id
AND  i.owner = 68
AND p.availability = 'true';
```

This simple query returns all the comments for the user with id 68. Before someone wants to bid a post, he may want to take a look at all the past comments for the owner of the post then this will be displayed.

```
SELECT c.*
FROM comment c, loan l, post p, item i
WHERE c.loan = l.loan_id
AND l.post = p.post_id
AND p.item = i.item_id
AND i.owner = 68;
```

This simple query returns all the bidding points, posts and bidding status of user with id 68. This will be displayed when a user wants to check his bidding history.

```
SELECT b.points, p.*,
CASE
WHEN loan_id IS NOT NULL THEN 'successful'
```

```
WHEN (loan_id IS NULL AND p.availability = ‘false') THEN 'fail'
ELSE 'pending' END AS bidding_status
FROM bid b
INNER JOIN post p ON (b.bidder = 68 AND b.post = p.post_id)
LEFT JOIN loan l ON (b.bidder = l.bidder AND b.post = l.post);
```

3.2 Aggregate Queries

This query returns all the user information and their average ratings. Sequence is ordered by average ratings from the highest to the lowest. This will be displayed when a someone wants to check whether a user has a good rating.

```
SELECT u.*, AVG(rating) as avg_rating
FROM users u
LEFT JOIN item i ON u.user_id  = i.owner
LEFT JOIN post p ON i.item_id = p.item
LEFT JOIN loan l ON p.post_id = l.post
LEFT JOIN comment c ON l.loan_id = c.loan
GROUP BY u.user_id, u.username, u.mobile, u.email, u.password, u.admin, u.points,
u.address
ORDER BY CASE WHEN AVG(rating) IS NULL THEN 0 ELSE 1 END DESC, avg_rating
DESC;
```

3.3 Nested Queries

This nested query returns all the unsuccessful bid together with the bidding points, corresponding postid and successful bidding points of this particular post. This will be displayed when a user wants to know whether his bid is failed.

```
SELECT ns.*, s.points AS successful_bidding_points
FROM (SELECT b.bidder, b.points, p.* FROM bid b
        INNER JOIN post p ON (b.post = p.post_id)
        LEFT JOIN loan l ON (b.bidder = l.bidder AND b.post = l.post)
        WHERE loan_id IS NULL) AS ns
INNER JOIN (SELECT b.bidder,b.points, b.post FROM bid b
                INNER JOIN loan l ON (b.bidder = l.bidder AND b.post = l.post)) AS s
ON ns.post_id = s.post
WHERE ns.bidder = 68;
```

3.4 Insertion, Deletion and Update

This query inserts an admin to the users table.

```
INSERT INTO users (user_id, username, mobile, email, password, admin) VALUES
(DEFAULT, 'cs2102', '3016779446', 'cs2102@nus.com', '123456789', true);
```

This query inserts a normal user into users table.
INSERT INTO users (user_id, username, mobile, email, password, admin) VALUES (DEFAULT, 'fbenninger1', '4345464198', 'mbarns1@mysql.com', 'UC2pBghtlp', false);

This query inserts an item into item table.
INSERT INTO item (item_id, name, owner, category, description) VALUES (DEFAULT, 'Book', 14, 'ALL', 'useful');

This query inserts a post into post table.
INSERT INTO post (post_id, title, item, start_time, end_time, description, delivery, status) VALUES (DEFAULT, 'useful book2', 1, '2018-08-26 16:25:22', '2018-11-10 18:56:11', 'Joq good', false, true);

This query inserts a bid into bid table.
INSERT INTO bid (bidder, points, post) VALUES (15, 1, 1);

This query inserts a loan into loan table.
INSERT INTO loan (loan_id, bidder, post, start_time, end_time) VALUES (DEFAULT, 2, 6, '2018-08-05 22:27:13', '2018-12-30 02:09:23');

This query inserts a comment into comment table.
INSERT INTO comment (comment_id, loan, user_name, content, rating) VALUES (DEFAULT, 1, 9, 'useful', 4);

This query happens when user 68 wants to delete his bid for post 7.
DELETE FROM bid
WHERE bidder = 68
AND post = 7;

This query deletes comment with id 7 from comment table.
DELETE FROM comment
WHERE comment_id = 7;

This query updates the bidding points of user 68 for post 7 to 200.
UPDATE bid
SET points = 200
WHERE bidder = 68 AND post = 7;

This query deducts 200 points from the total points of users 68.
UPDATE users
SET points = points-200
WHERE userid = 68;

This query updates the availability of post with id 68 to ‘false’.
UPDATE post

```
SET availability = 'false'
WHERE post_id = 68;
```

3.5 Trigger

The following function trigger is to check before each insertion into post, whether there is an available post about the same item. If there is such post, the insertion won't be conducted and an error message will be printed.

```
CREATE OR REPLACE FUNCTION check_error()
RETURNS TRIGGER AS $$
BEGIN
IF (SELECT count(*)
        FROM post p
        WHERE p.item = NEW.item
        AND p.availability = 'True')  >= 1
Then RETURN CAST('This item is being posted now.' as int);
ELSE
RETURN NEW;
END IF;
END; $$ LANGUAGE PLPGSQL;

CREATE TRIGGER post_insert
BEFORE INSERT
ON post
FOR EACH ROW
EXECUTE PROCEDURE check_error();
```

The following function trigger is to check before each insertion into post, whether the number of available posts the user owns exceeds 6. If he/she already has 6 available posts, the insertion won't be conducted and an error message will be printed.

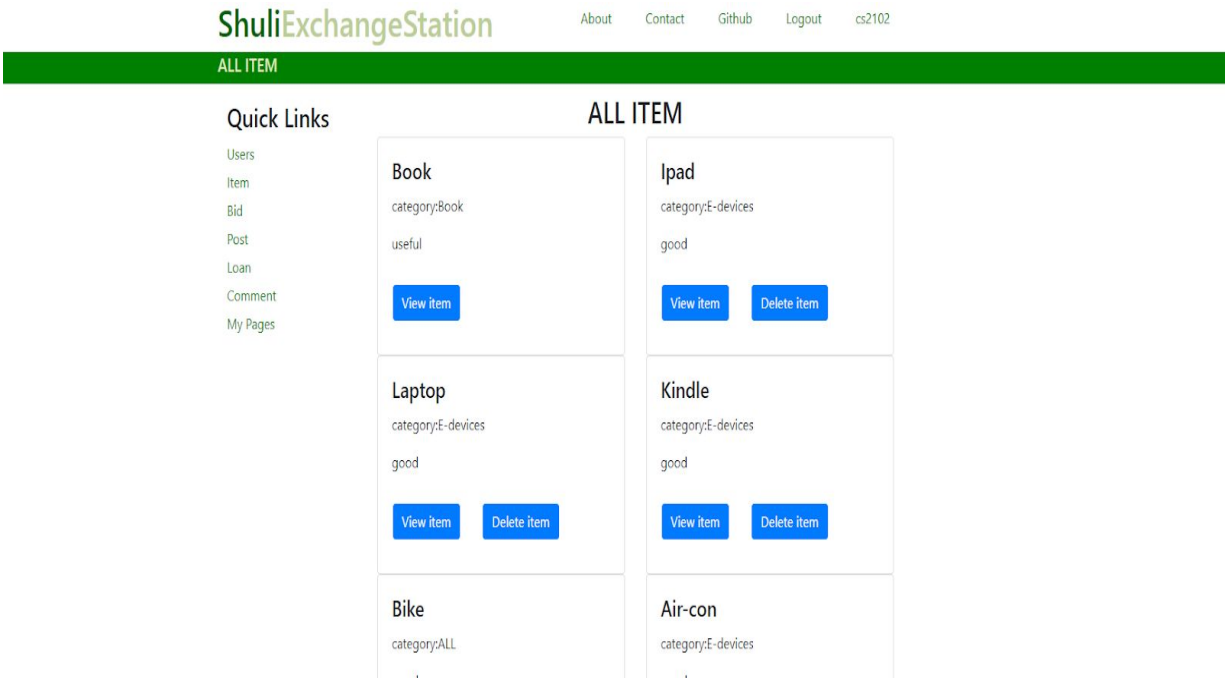
```
CREATE OR REPLACE FUNCTION user_post_error()
RETURNS TRIGGER AS $$
BEGIN
IF (SELECT count(*)
        FROM post p, item i, users u
        WHERE p.item = i.item_id
        AND i.owner = u.user_id
        AND p.availability = 'True'
        AND u.user_id = (SELECT i2.owner FROM item i2 WHERE NEW.item = i2.item_id)
        GROUP BY u.user_id)  >= 10
Then RETURN CAST('You can only have at most 10 available posts in total.' as int);
ELSE
RETURN NEW;
END IF;
```

```
END; $$ LANGUAGE PLPGSQL;

CREATE TRIGGER too_many_post
BEFORE INSERT
ON post
FOR EACH ROW
EXECUTE PROCEDURE user_post_error();
```

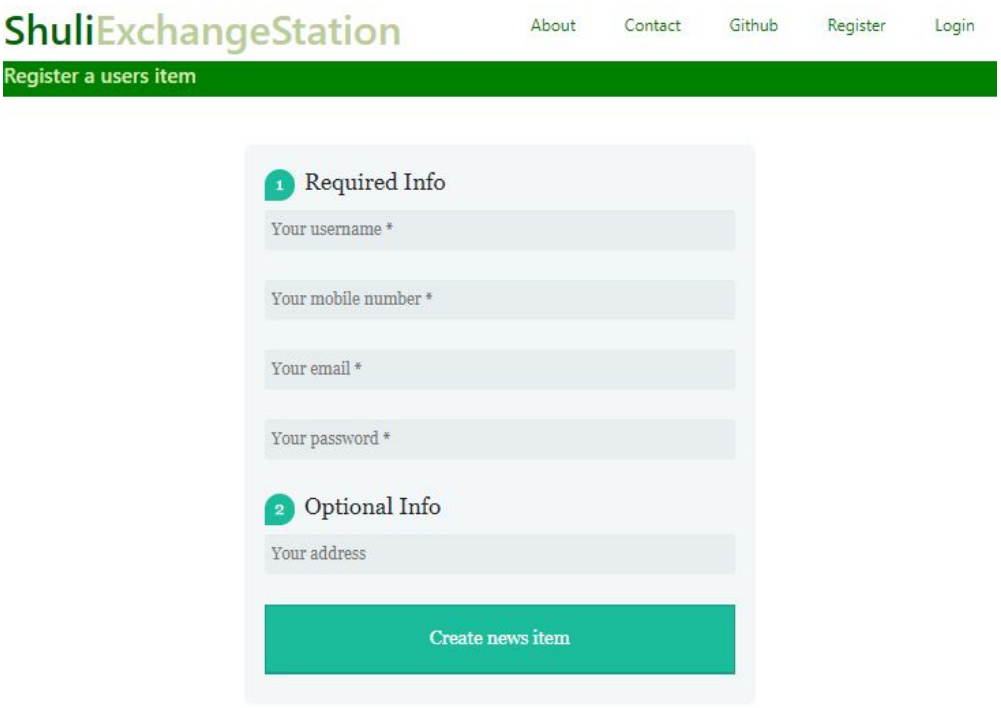
4. Website

This website is a database web app developed with PHP. After a research on the available PHP MVC frameworks, we chose Codeigniter framework to implement this website. The user interface is designed by us with the help of several open source HTML templates.



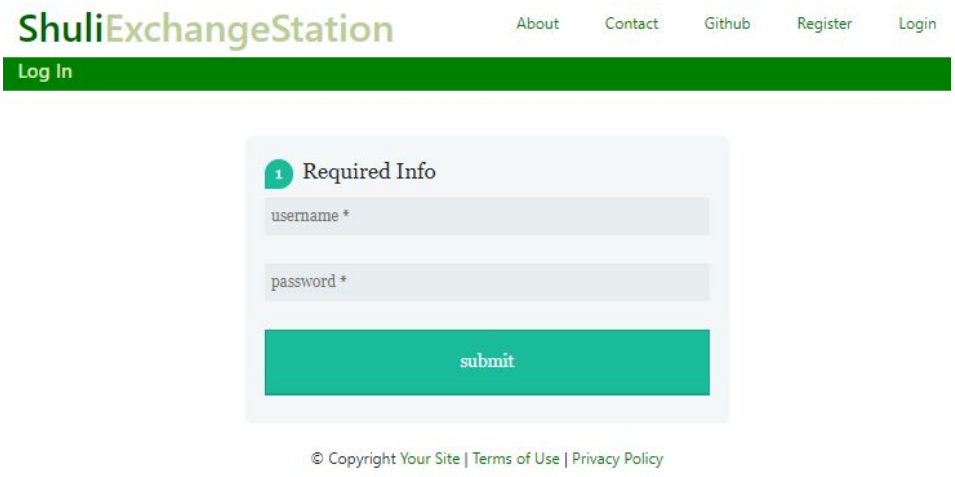
4.1 Sign up

After user clicks the “Register” button at top right corner of home page, he will be redirected to the sign up page so that he can input his username, password etc. to create a new account.



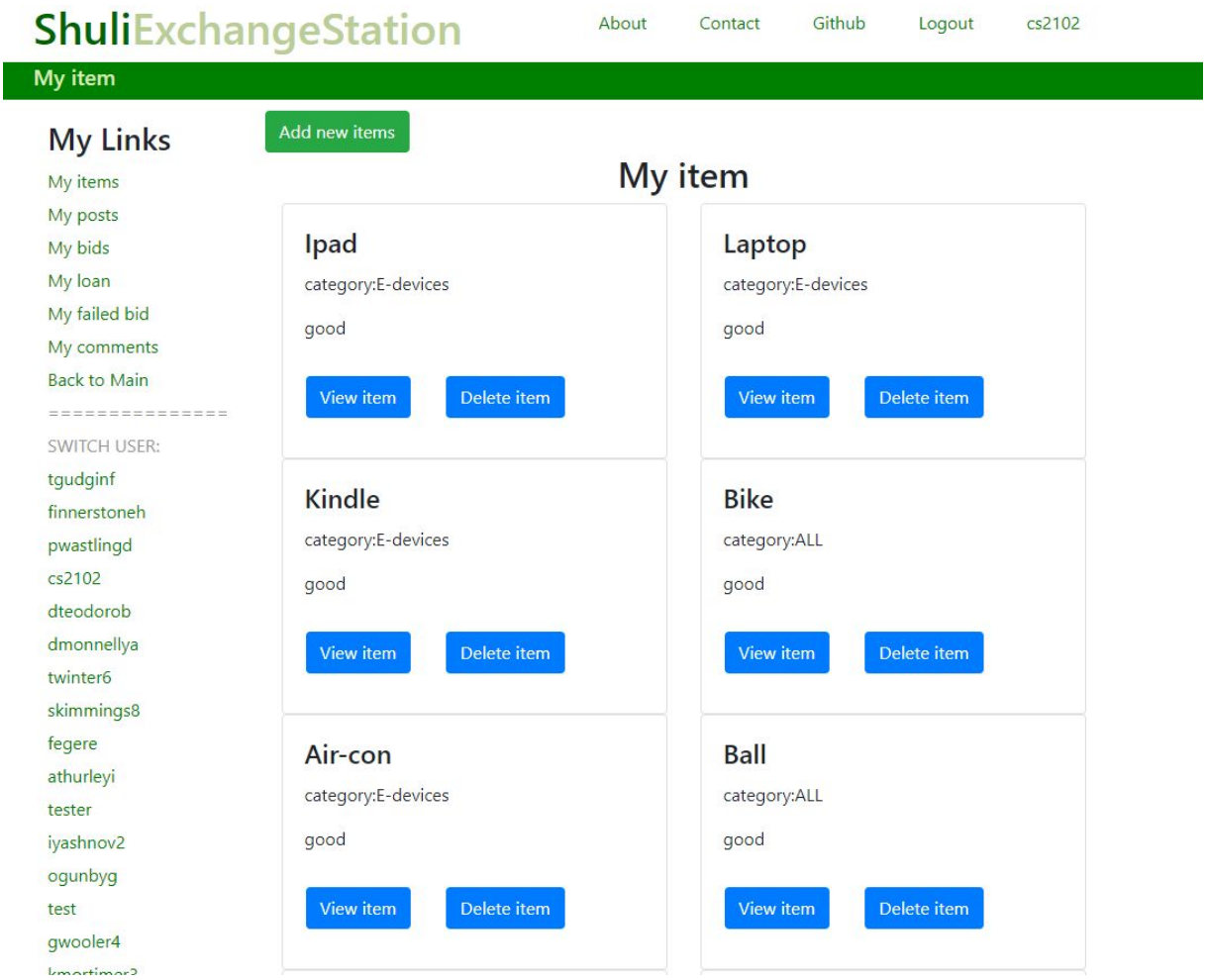
4.2 Login

Clicking the “Login” button allows users to login to their own accounts.



4.3 User View

The following page show the admin user's view. Under "Back to Main", there is a list of normal usernames. An admin can click any username and switch to their account, so that he can manage any account, item, post, bid, comment or loan.



The only difference between the view of normal user and admin user is that there is no "Switch to User" option for normal users. The buttons under "My Links" allow users to view and manage their own items, posts, bids ,loans and comments. "My bid" shows all the pending bid of the user. "My failed bid" shows all the unsuccessful bids of the user. "My loan" shows all the successful bids together with loans of the user.

4.4 Overall workflow

To sum up, our database web app allows users to create new accounts, add new items, create new posts, bid for their interested posts and also accept others' bid for their posts. On our website, which bid to accept is totally decided by the owner of the post. Once a bid is

successful, user can borrow the item mentioned in the post from the owner which means a loan is created and the corresponding bidding points will be deducted from their accounts. Also once a loan was created, the availability of this post will become ‘false’ and users cannot bid this post any more.