CS2102 Database
Project Report

Carolend

DONG SHAOCONG, HE XINYI, LIU YULIN, WANG ZEXIN

Department of Mathematics

National University of Singapore

AY2017/18 Semester 1

Abstract

Acknowledge

We would like to thank Associate Professor Bressan Stephane for his helpful supervision throughout the course of this project.

Contents

1 Introduction

In this project, we are required to build a stuff sharing website. 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, where to pick up and return, 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. Administrators can create, modify and delete all entries.

1.1 Developing Specifications

After seeing through the relevant products specifications and the website requirements, we decided to use PHP as back end programming language, $Javascript$ as front end developing language, $MySQL$ as our database. We used $Laravel$, which is the most popular web application framework for PHP.

From the project requirement, Eloquent ORM in built in Laravel to access and manipulate the database is not allowed. Therefore, anything related to database management, access, and manipulation is down by importing PHP $mysqli$ library and execute raw SQL queries.

To develop this web application, we utilise the Model View Controller (MVC) framework.

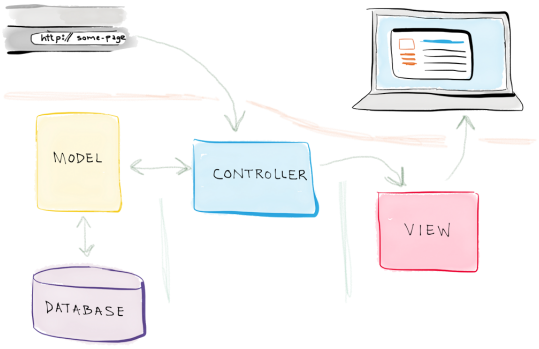

Figure 1: Model View Controller (MVC)

We use CSS Scaffolding in Laravel to take care of the views. Buttons and redirections from the user side are implemented in $Javascript$. The models and controllers are written in PHP. The specific $MySQL$ database we used it $root$ user's database named $blog$. The website is still hosted on `localhost:8000` only.

From our modelling for this problem, there are two types of users. The admin users will have different interfaces and unlimited access to all entries. From the user management button in the admin user's profile page. All the users information can be modified and deleted. New users can be added on this panel as well.

2 Database Design

2.1 Entity-Relationship Diagram

2.2 Entities

Users:

Attribute	Domain
email	VARCHAR(64)
username	VARCHAR(64)
password	VARCHAR(64)
mobile	INT(8)
address	VARCHAR(128)
points_available	INT(3)
admin	INT(1)
credit_rating	NUMERIC
created_at	TIMESTAMP

Items:

Attribute	Domain
name	VARCHAR(64)
avatar	VARCHAR(256)
owner	VARCHAR(64)
description	TEXT
available	VARCHAR(5)
created_at	TIMESTAMP

Posts:

Attribute	Domain
item	VARCHAR(64)
title	VARCHAR(64)
location	VARCHAR(128)
description	TEXT
start	TIMESTAMP
end	TIMESTAMP
created_at	TIMESTAMP

Bids:

Attribute	Domain
bidder	VARCHAR(64)
post	VARCHAR(64)
status	CHAR(7)
points	INT(3)
created_at	TIMESTAMP

Loans:

Attribute	Domain
bid	VARCHAR(64)
post	VARCHAR(64)
start	TIMESTAMP
end	TIMESTAMP
comments	TEXT
status	VARCHAR(8)
created_at	TIMESTAMP

2.3 Relational Schema

CREATE TABLE users (
email VARCHAR(64) PRIMARY KEY,
username VARCHAR(64) NOT NULL,
password VARCHAR(64) NOT NULL,
mobile INT NOT NULL,
address VARCHAR(128),
points_available INT DEFAULT 500,
admin INT DEFAULT 0,
credit_rating NUMERIC CHECK (credit >= 0 AND credit <= 5),
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE items (
itemid INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(64) NOT NULL,
avatar VARCHAR(256),
owner VARCHAR(256) REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE,
description TEXT,
available VARCHAR(5) CHECK(available = 'TRUE' OR available = 'FALSE'),
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE posts (
postid int AUTO_INCREMENT PRIMARY KEY,
item int REFERENCES items(itemid) ON DELETE CASCADE,
start TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
end TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
title VARCHAR(64) NOT NULL,
location VARCHAR(128) NOT NULL,
description TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
CHECK (start < end)
);

CREATE TABLE bids (
bidid INT AUTO_INCREMENT PRIMARY KEY,
bidder INT REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE,
post INT REFERENCES posts(postid) ON DELETE CASCADE,
points INT NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
status CHAR(7) CHECK (status = 'SUCCESS' OR status = 'FAILURE')

);

CREATE TABLE loans (
loanid INT AUTO_INCREMENT PRIMARY KEY,
bid INT REFERENCES bids(bidid) ON DELETE CASCADE,
post INT REFERENCES posts(postid) ON DELETE CASCADE,
start TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
end TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
comments TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
status VARCHAR(8) CHECK (status = 'ONLOAN' or status = 'RETURNED' or status = 'EXPIRED'),
CHECK (start < end)
);

2.4 Schema Functions

3 SQL Queries

3.1 Simple Queries

The following simple query returns all the information about one particular bidding specified by bidid. This is displayed when the bidder wants to edit his own bidding.

SELECT *
FROM bids
WHERE bids.bidid = bidid;

The following simple query returns all the information about all the items. This is displayed when user accesses the 'items' page.
SELECT *
FROM item i;

The following simple query returns all the information about one particular post. This is displayed when the poster wants to edit his own post.
SELECT *
FROM posts
WHERE posts.postid = postid;

The following simple query returns all the information about one user with userid specified. This is displayed when the user visited his 'user' page.
SELECT *
FROM users
WHERE users.id = userid;

The following simple query returns all the information about the posts of a particular user specified by his/her email.

SELECT *
FROM posts p, items i
WHERE p.item = i.itemid AND i.owner = email;

The following simple query returns all the information about the transaction history between two users, i.e. their usernames, bidding points, bidding time.

SELECT u1.username as owner, u2.username as bidder, b.points, b.updated_at as time
FROM users u1, users u2, bids b, items i, posts p
WHERE b.bidder = u2.email AND b.post = p.postid AND p.item = i.itemid AND i.owner = u1.email
AND (b.bidder = email OR i.owner = email);

3.2 Aggregate Queries

The following aggregate query returns the maximum bidding points for one particular post with postid specified. This is displayed when the poster wants to check the maximum bid made for his post.

SELECT MAX(b.points)
FROM bids b, posts p
WHERE b.post = p.postid AND p.postid = postid;

3.3 Nested Queries

The following nested query SELECT p.postid, p.title, p.description, p.created_at, i1.avatar
FROM posts p, items i1
WHERE p.item = i1.itemid
AND p.item NOT IN
(SELECT i.itemid FROM items i WHERE i.owner = email);

3.4 Queries using INNER JOIN

3.5 Queries using EXISTS

3.6 Queries using set operations

3.7 Insertions, Deletions and Updates

The following query inserts one bidding record into the bids table.

INSERT INTO bids (bidder, post, points) VALUES (email, postid, points);

The following query inserts one loan record into the loans table.

INSERT INTO loans (bid, post, status) VALUES(bidid, postid, using_status);

The following query inserts one item record into the items table.

INSERT INTO items (description, available, name, owner, avatar) VALUES (description, available, name, owner, filename);

The following query inserts one post record into the posts table.

INSERT INTO posts (item, title, location, description) VALUES (itemid, title, location, description);

The following query inserts one unsuccessful bidding record into the bids table.

INSERT INTO bids (status, bidder, post, points) VALUES (‘FAILURE’, email, postid, point);

The following query deletes one bidding record.

DELETE FROM bids
WHERE bids.bidid = bidid;

The following query deletes one post record.

DELETE from posts
WHERE posts.postid = postid;

The following query updates the bidding points for one particular bidding.

UPDATE bids
SET bids.points = points_updated
WHERE bids.bidid = bidid;

The following query sets the bidding status to success

UPDATE bids
SET bids.status = ’SUCCESS’
WHERE bids.bidid = bidid;
UPDATE users set users.points_available = users.points_available + points where users.id = userid;

The following query updates the loan information to be ‘returned’ after the borrowed item has been returned to the owner.

UPDATE loans l
SET l.status = ‘RETURNED’
WHERE l.loanid = loanid;

The following query updates the post information of a particular post specified by postid.

UPDATE posts
SET posts.title = title, posts.location = location, posts.description = description
WHERE posts.postid = postid;

3.8 Query using view

Create a view item_popularity which contains all the popularity information about the items.

CREATE VIEW item_popularity AS

```
SELECT i.itemid as itemid, i.owner as owner, COUNT(*) AS popularity
FROM items i, posts p, bids b
WHERE i.itemid = p.item AND p.postid = b.post
GROUP BY i.itemid, i.owner;
```

The following query selects the average popularity of the items posted by each user.

```
SELECT u.email, AVERAGE(i.popularity)
FROM users u, item_popularity i
WHERE u.email = i.owner
GROUP BY u.email;
```

The following query selects the average popularity of the items bided by each user.

```
SELECT u.email, AVERAGE(i.popularity)
FROM users u, item_popularity i, posts p, bids b, loans l
WHERE u.email = b.bidder AND b.bidid = l.bid AND b.post = p.postid AND p.item = i.itemid
GROUP BY u.email;
```

The following query remove the view created.

```
DROP VIEW if exists item_popularity
```

4 Web Interface Design

We aims to have a friendly, easy-to-use interface for our web application. The following sections are about the pages in our web interface.

4.1 Sign Up Page

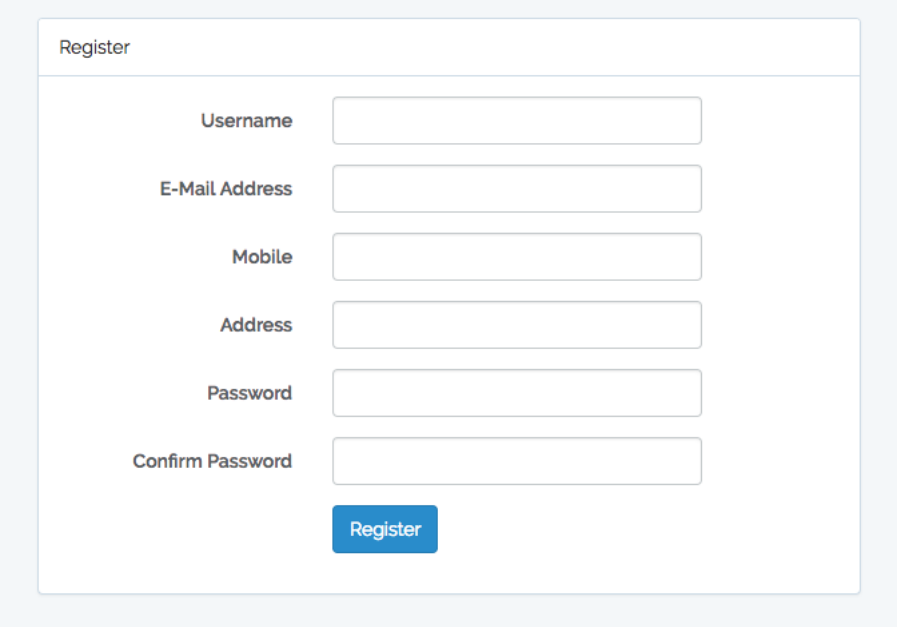

Figure 2: New User registration page

From this page, new users will be able to sign up and enter our system.

4.2 Log In Page

Other users will be able to log in via this page.

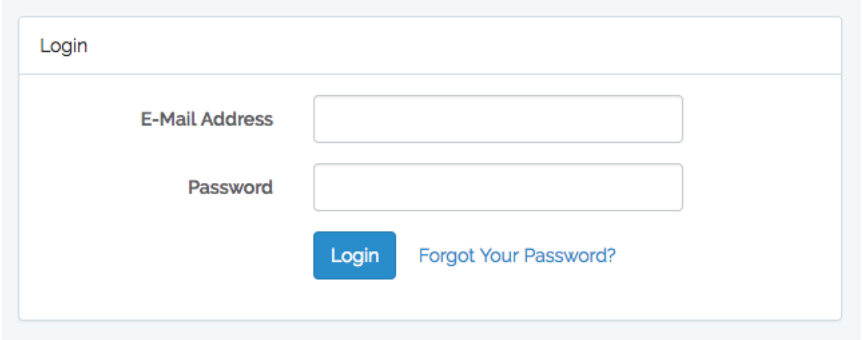

Figure 3: User login page

4.3 Application

Upon logging in, the user will be directed to his/her dashboard. He can easily see the points available for him or her. Besides that, the items the user owning, the posts he or she has submitted, the posts the user is currently bidding for and the items the user is currently borrowing will be displayed sequentially throughout this the page.

In addition, from the top right drop down list, the user can go to other pages ranging form his

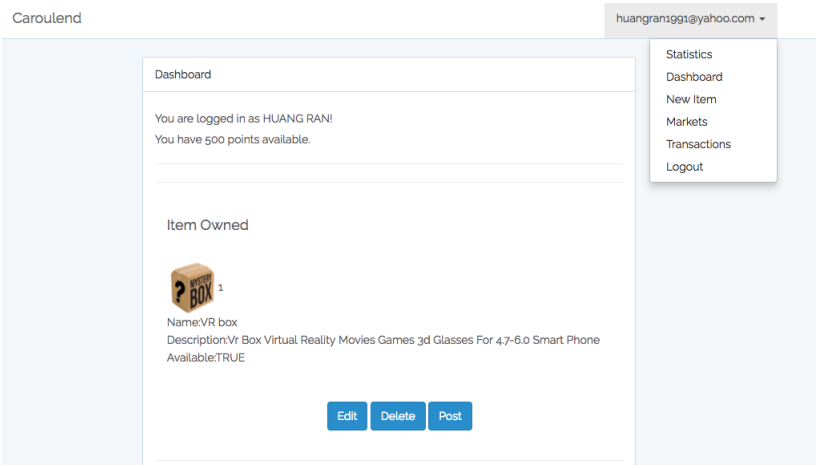

Figure 4: User login page

or her using statistics. User can also create new item, go to the posting markets and checkout current transactions and transacting history. This includes the user's posts that are being bid by other users, the user's current lending items and the past transactions.

4.4 Administration

There is a special type of user called admin user. They have their admin column equal to 1 and the can have unlimited access to access, delete, update and create any entries in our database. Users are not able to sign up an admin account. The only possible way to become an admin is to update the table in our database directly. Implemented in this way, the administration page is very secure and the current normal users' information are properly protected. As shown in this screen shot,

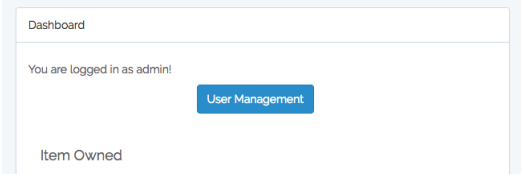

Figure 5: User login page

the admin user can access any entries inside this application. If he or she wants to manage the current users, they can do so by clicking the user management button and the user records can be updated, deleted or created on our administration user management panel.

5 Conclusion

correct way of citing something:[1]

References

[1] Yves Hilpisch, *Python for Finance*. O'Reilly Media, 2015.

[2] Paul Glasserman, *Monte Carlo methods in financial engineering*. Springer, 2010.

[3] Daniel Duffy, *Finite difference methods in financial engineering: A partial differential equation approach*. John Wiley&Sons, 2006.

[4] Mark Broadie, Paul Glasserman, Steven Kou, *A Continuity Correction for Discrete Barrier Options*. Mathematical Finance, 1997.