CIS 550 Final Project Report: PSQUARE

Group 5
Kuan Ting Chen, Danyang Cong, Alen Kubati, Seth Shannin

1. Introduction

The task for this project was to create a social photo-sharing application. We have successfully implemented a wide array of basic functionalities a modern user would expect to see in such a social networking site. We have named our service "PSQUARE."

Specific goals and functionalities we wanted to tackle at the beginning of the project are summarized below:

- Data integration capability
- Authentication system
- Registration system: new users can register to use PSQUARE
- User profile: a user has his personal information displayed
- Friendship management: a user can see his own friends, group friends into circles, add friends, and get friend recommendations
- Friendship visualization: provide an easy and intuitive interface for users to explore their relationship network
- Photo: a user can post, search, rate, and tag photos
- Update feeds: a user is notified or recent activity by his friend

We are pleased to note that we successfully completed all the base requirements we laid out in the list above. In addition, we added the following features:

- Embedded a live chat service in the site for PSQUARE users to communicate with each other while browsing
- Added stemming support to photo search functionality

2. System Overview and Architecture

PSQUARE is a complex system that incorporates different programming tools and various data models. This section describes the architecture of the system.

2.1 Software Stack

We implemented PSQUARE mainly with the Jetty web server and Google Web Toolkit (GWT). GWT is a good web development toolkit for developing complex JavaScript and AJAX browser-based applications. Since PSQUARE is a social networking application and high interactivity is assumed, the choice of GWT was a very natural one. Eclipse has plugins for GWT and for many of the other tools we used; this system eased setup effort and allowed for smoother collaboration between team members. For persistence of data, we used MySQL. MySQL is a high-performance database which is utilized in many popular websites including Twitter, Flicker, Facebook, and Wikipedia. Our system also used Zorba for XQuery programs, mainly for data integration. PSQUARE can successfully integrate other groups' XML export through our data translation program (written in

XQuery). We also used Python for intermediary data processing such as data translation (see **section 3.1** for more details) and background tasks.

2.2 Data Model

Data models in PSQUARE can be divided into two categories: XML and SQL.

2.2.1 XML

The system must deal with XML-formatted data in two different aspects. The first concerns integration of data from other groups. To convert heterogeneous data sources, we first convert their data into a simple common schema. This helps account of the differences in the XML schema between different groups. Once the raw XML sources are converted to our common schema, we take the simplified representation and flatten it into tuples. These are then inserted into the database.

The common schema is designed to mimic our SQL schema field-for-field. That is, we deliberately avoid nesting any elements. This makes it easy to convert between the common XML format and tuples for the database. The following is an example of what our data translation programs try to capture for the USER type in a given XML file:

```
<USER>
   <ID>{data($x/tns:accountID)}</ID>
   <FNAME>{data($x/tns:fname)}</FNAME>
   <LNAME>{data($x/tns:lname)}</LNAME>
   <BDAY>{data($x/tns:birthDate)}</BDAY>
   <EMAIL>{data($x/tns:email)}</EMAIL>
   <GENDER>{data($x/tns:gender)}</GENDER>
   <ADDRESS>{data($x/tns:address)}</ADDRESS>
   <PW>{data($x/tns:password)}</PW>
</USER>
```

For the other types, please see **Appendix** for more details.

The second place where XML data model shows up is the friendship browser. PSQUARE generates XML files for the friendship browser using data from the MySQL database. The friendship browser uses the generated information from the XML files to display the visualization (the way the friendship browser works is very similar to HW4).

2.2.2 SQL

There are 13 MySQL tables in the system. These tables include Users, Interest, AcademicInstitution, Professor, Student, Circle, FriendRelation, PrivacyType, Rating, Tag, UserPrivacy, and CirclePrivacy. See **Appendix** for the details of the SQL DDL.

2.3 System Diagram

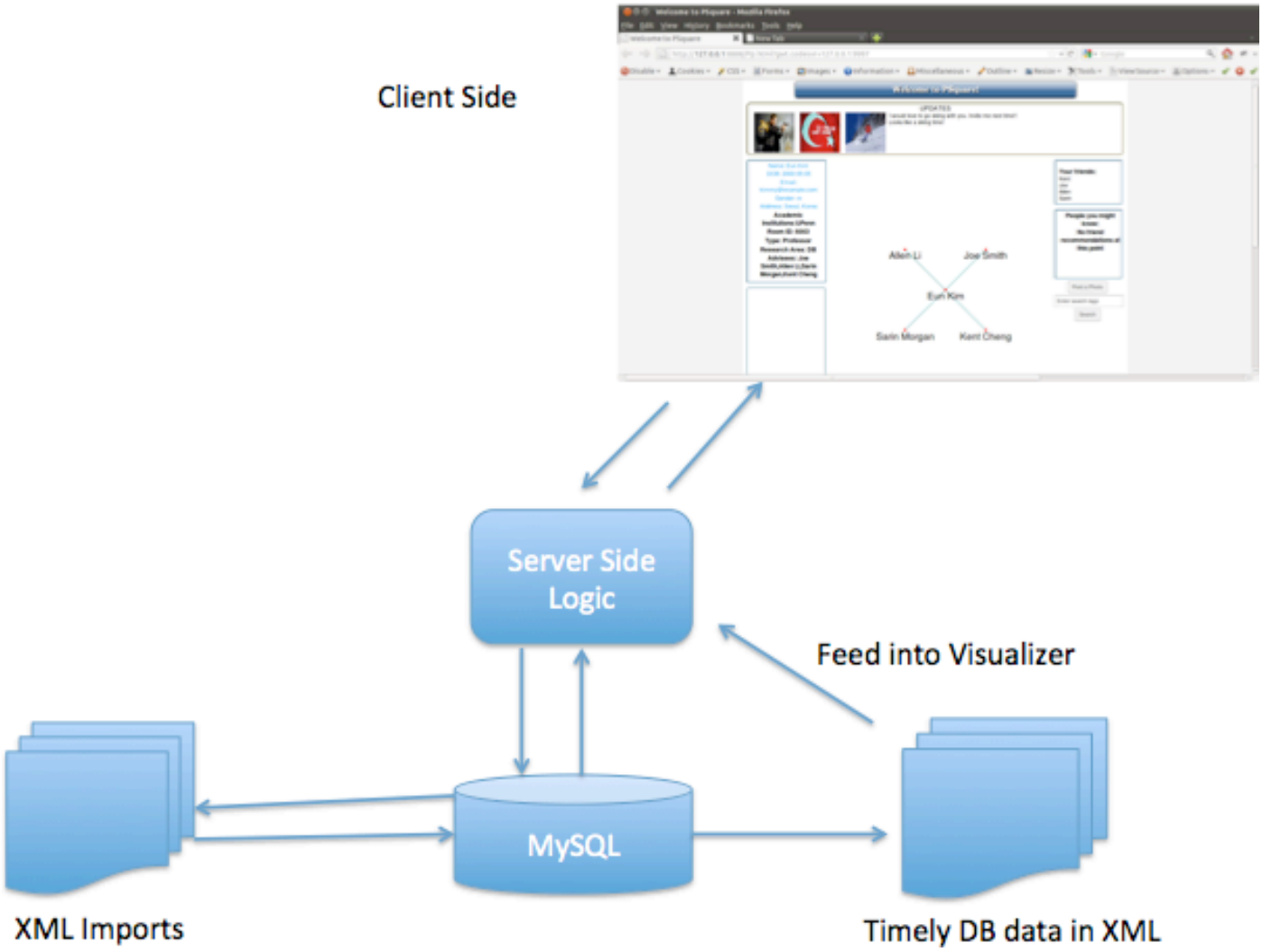

Fig. 2-1 A schematic drawing of PSQUARE system

3. Design and Implementation

Throughout the development process, many design decisions and implementation strategies have been planned. This section documents the design behind each piece of important functionality and the rationale behind choosing a particular approach.

3.1 Importing Data From Other Groups

We use XQuery routines to transform the XML data from each other group into the common format described above in *Section 2.2.1*. The translated data is then flattened into tuples and stored into the MySQL database by a Python program. There is a small shell script to automate this process.

Some issues we ran into and the way we solved them:

- Groups who had age but not birthday information for users
    - We assigned these users a Jan. 1 birthday for the correct year
- XQuery and MySQL represent dates in different formats

    - Dealt with this when converting from common XML format into SQL tuples using Python string magic
- Groups allowed anonymous tags
    - We added a blank user with `userid 0` and assigned all anonymous tags to this user
- One group did not enforce the uniqueness of Circles
    - We create a (hopefully) unique name by concatenating the userid and the hash of the circle's name
- Every group had a different way of representing photo privacy
    - This was simply dealt with on a case-by-case basis, converting into the common format.

Please see the "`pp/war/psquare_data`" directory for relevant files.

3.2 Login System

Access is only granted to users who pass authentication. The front-end system reads the email address and password. It then verifies with the database to make sure the credentials are valid.

3.3 User Profile

One of the most basic elements of the system is simply displaying the user's personal information to them. This includes address, birth date, advisor info, etc.

3.4 Friendship System

PSQUARE goes beyond what was required and supports new-user registration. A user can also add new friends through the friend recommendation list. When adding new friend, user can assign the friend to one or more circles.

We recommend new friends to users if they have a mutual friend or if the have rated the same photo.

3.5 Friendship Browser

The Friendship Browser uses the JavaScript InfoVis Toolkit (JIT) for visual effects. It extracts information from an XML file which is in turn generated by a small Python program. This Python program is periodically invoked in its own thread on the sever, ensuring that the XML representation of the data is kept up-to-date. A user can browse up to 2 hops in the Friendship Browser (can see friends and friends of friends) as per requirement.

3.6 Circles

PSQUARE supports the basic circle requirements. Every friend of some user must be in at least one of that users circles (in fact, this is how we define friendship). When adding new friends, one can of course designate to which circles they should be added. Also, when uploading new photos, users can specify which circles the photo should be visible to.

3.7 Photo Posting

A user can post new photos by supplying a URL. In addition, users may specify tag words as they are posting. These tags then become associated with the photo and are

used to help other users search for photos. More tags can of course be added at a later date (either by the original poster or by anyone else to whom the photo is visible).

Accounting for human errors, PSQUARE also has a confirmation window which pops up after the user presses `Upload` The confirmation windows contains a preview of the photo and makes the user confirm his/her choice to post it. Once the user confirms, he or she can customize the visibility of the photo.

3.8 Photo Tagging and Rating

Any user-added photo, when displayed anywhere on the site, is associated with a click handler. This allows users to tag and rate any photo that appears on the site simply by clicking it.

3.9 Photo Search

PSQUARE supports searching photos for photos by tag words. PSQUARE uses the Porter Stemmer algorithm for searching. The system maintains an inverted index table, which allows for looking up the photos by keyword. This inverted index is periodically updated by a server-side background thread, ensuring that the search results are relatively up-to-date.

3.10 Photo Relevance Scoring

Every photo is assigned a relevance score per user as follows:

```
score = 0;
if( areFriends(this_user, photo_owner)
   score += 4.51;
score += averageRatingForPhoto();
score += 1.01 * averageRatingForPhotoByThisUsersFriends();
```

In words, a photo is owned by a friend, a receives a boost of 4.51 points. The average rating of the photo then added on. Finally, we add on the average rating by friends, multipled by a factor of 1.01. We note that the decimal components of this scoring system may seem rather arbitrary, and in fact they are. The decimal components are used to give preference to one attribute over another in what would otherwise be tied scores.

Since these scores are relatively expensive to compute, we implemented a caching layer on top of this. Top scoring photos are stored per user. There is a timeout associated with each entry to prevent information from becoming too stale.

3.11 Update Feeds

Our system fortunately provides a simple way to track recent updates. The way we have implemented photo uploading ensures that each new photo added always get the next highest id. That means that finding recent photos equates to just finding the photos with the highest id.

In the schema of our system, we find recent photos as follows:

```
SELECT P.pid, P.url
FROM Photo P
```

```
WHERE P.privacy='ALL'
   OR %USER in (SELECT UP.userid
                FROM UserPrivacy UP
                WHERE UP.pid=P.pid)
   OR %USER in (SELECT FR.friendid
                FROM CirclePrivacy CP, FriendRelation FR
               WHERE CP.pid=P.pid AND CP.cid=FR.cid)
ORDER BY P.pid DESC
LIMIT TRENDING_PHOTO_AMOUNT;
```

3.12 Chat Server

We found it very simple to implement a chat system. Users simply publish messages to the server, which keeps the messages in a list based on the order it receives them. As users log on, an RPC call is made to find the current message number. After this, the client then periodically polls for messages numbered higher than the last message it knows about.

We debated storing these chats in a database instead of just a memory-based list. We decided however that there was no huge advantage to having chats persist (in that users likely do not expect chats to be logged across sessions).

4. Division of Labor

The following is a list of features/functionalities each individual in our group worked on:

- Kuan Ting Chen: login system, user profiles, photo posting, photo tagging and rating, photo search, general integration tasks, report
- Danyang Cong: registration, photo posting, photo search, photo tagging and rating, update display, friending, layout
- Alen Kubati: friendship browser, friend recommendation, photo search, friend info and recommendation display, friend photo display.
- Seth Shannin: Site layout, chat server, XML import/export, inverted index for tag search, stemming support for searches, friend recommendations, photo relevance scoring, general integration tasks.

5. Appendices

5.1 Appendix A: Selected XQuery Code

5.1.1 Circle hashing function (for circle uniqueness)

```
declare namespace tns = "http://www.example.org/pennphoto";
declare namespace functx = "http://www.functx.com";

import module namespace hash "http://www.zorba-xquery.com/modules/cryptography/hash";

declare function local:hashcircle($id as element(tns:id), $c as element(tns:circle))
      as xs:string {
      (: xs:string(data($id)) + :)
```

```
      let $joined := concat(xs:string(data($id)), hash:md5(data($c/tns:name)))
      let $sanitized := replace($joined, 'a|b|c|d|e|f', '')
      return fn:substring($sanitized, 0, 10)
};
```

5.1.2 Converting user info into common XML format

```
for $x in doc('pennphoto-18.xml')/tns:photodb/*
return
      <USER>
      <ID>{data($x/tns:id)}</ID>
      <FNAME>{data($x/tns:first_name)}</FNAME>
      <LNAME>{data($x/tns:last_name)}</LNAME>
      <BDAY>{year-from-date(current-date()) - data($x/tns:age)}</BDAY>
      <EMAIL>{data($x/tns:email)}</EMAIL>
      <GENDER>{data($x/tns:gender)}</GENDER>
      <ADDRESS></ADDRESS>
      <PW>{data($x/tns:password)}</PW>
      </USER>
}
```

5.1.3 Converting photos into common XML format

```
for $x in doc('pennphoto-12.xml')/tns:photodb/*
return
      {
      for $y in $x/tns:photo
      return
      <PHOTO>
            <PID>{data($y/tns:photoID)}</PID>
            <USERID>{data($x/tns:accountID)}</USERID>
            <URL>{data($y/tns:url)}</URL>
            <PRIVACY>{if (data($y/tns:visibility) = 0) then "ALL" else
"CUSTOM"}</PRIVACY>
      </PHOTO>
      }
```

5.1.3 Coverting circle-based photo privacy into common XML format

```
for $x in doc('pennphoto-22.xml')/tns:photodb/*
return
      {
      for $y in $x/tns:photo
      return
            {
            for $z in $y/tns:visibility/tns:circleID
            return
            <CIRCLEPRIV>
                  <PID>{data($y/tns:photoID)}</PID>
                  <CID>{data($z)}</CID>
            </CIRCLEPRIV>
            }
      }
```

5.2 Appendix B: MySQL DDL

```
CREATE TABLE `Users`(
        `userid` int(11) NOT NULL auto_increment,
        `first_name` varchar(20),
        `last_name` varchar(20),
        `dob` date,
        `email` varchar(40),
```

```
        `gender` char(1),
        `address` varchar(100),
        `password` varchar(20),
        PRIMARY KEY (userid)
) ENGINE=InnoDB;

CREATE TABLE Interest(
        `userid` int(11) NOT NULL,
        `interest` varchar(40) NOT NULL,
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (userid, interest)
) ENGINE=InnoDB;

CREATE TABLE AcademicInstitution(
        `userid` int(11) NOT NULL,
        `institution` varchar(50) NOT NULL,
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (userid, institution)
) ENGINE=InnoDB;

CREATE TABLE Professor(
        `userid` int(11) NOT NULL,
        `research_area` varchar(20),
        `room_id` varchar(20),
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (userid)
) ENGINE=InnoDB;

CREATE TABLE Student(
        `userid` int(11) NOT NULL,
        `advisorid` int(11),
        `year` int(11),
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        FOREIGN KEY (advisorid) REFERENCES Professor(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (userid)
) ENGINE=InnoDB;

CREATE TABLE Circle(
        `cid` int(11) NOT NULL,
        `userid` int(11) NOT NULL,
        `name` varchar(20) NOT NULL,
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (cid)
) ENGINE=InnoDB;

CREATE TABLE FriendRelation(
        `friendid` int(11) NOT NULL,
        `cid` int(11) NOT NULL,
        FOREIGN KEY (friendid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        FOREIGN KEY (cid) REFERENCES Circle(cid)
                ON DELETE CASCADE,
        PRIMARY KEY (friendid, cid)
) ENGINE=InnoDB;

CREATE TABLE PrivacyType(
       `privacy` varchar(20) NOT NULL,
       PRIMARY KEY (privacy)
) ENGINE=InnoDB;

CREATE TABLE Photo(
          `pid` int(11) NOT NULL,
         `userid` int(11) NOT NULL,
         `url` varchar(500) NOT NULL,
```

```
        `privacy` varchar(20) NOT NULL DEFAULT 'ALL',
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        FOREIGN KEY (privacy) REFERENCES PrivacyType(privacy),
        PRIMARY KEY (pid)
) ENGINE=InnoDB;

CREATE TABLE Rating(
        `pid` int(11) NOT NULL,
        `userid` int(11) NOT NULL,
        `score` int(11) NOT NULL,
        FOREIGN KEY (pid) REFERENCES Photo(pid)
                ON DELETE CASCADE,

  FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (pid, userid, score)
) ENGINE=InnoDB;

CREATE TABLE Tag(
        `pid` int(11) NOT NULL,
        `userid` int(11) NOT NULL,
        `comments` varchar(250) NOT NULL,
        FOREIGN KEY (pid) REFERENCES Photo(pid)
                ON DELETE CASCADE,
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (pid, userid, comments)
) ENGINE=InnoDB;
CREATE TABLE UserPrivacy(
        `pid` int(11) NOT NULL,
        `userid` int(11) NOT NULL,
        FOREIGN KEY (pid) REFERENCES Photo(pid)
                ON DELETE CASCADE,
        FOREIGN KEY (userid) REFERENCES Users(userid)
                ON DELETE CASCADE,
        PRIMARY KEY (pid, userid)
) ENGINE=InnoDB;

CREATE TABLE CirclePrivacy(
        `pid` int(11) NOT NULL,
        `cid` int(11) NOT NULL,
        FOREIGN KEY (pid) REFERENCES Photo(pid)
                ON DELETE CASCADE,
        FOREIGN KEY (cid) REFERENCES Circle(cid)
                ON DELETE CASCADE,
        PRIMARY KEY (pid, cid)
) ENGINE=InnoDB;
```