6083 Principles of Database Systems

project 2 Oingo

Team member:

Yiqiu Wu (yw3101)

1. Project Introduction

Introduction:

This project's name is Oingo. This app is meant to provide a platform for users to share their notes, recommendations or remind themselves. The main function is that users can post notes with location, time stamps and other customized information. Others will receive these notes only if these notes meet all the conditions. Of course, users can choose what kind of notes they can receive with bunch of filters.

The whole project is coded with php, and is realized on web pages by WAMP(a software stack for the Microsoft Windows operating system, consisting of the Apache web server, OpenSSL for SSL support, MySQL database and PHP programming language).

Both the SQL files which containing creating and inserting data are attached with the project. If you want to run the project in your own computer, make sure that your WAMP environment are installed correctly, and the program can access localhost.

Explanations:

Users can access Oingo through web pages. Different users can create their own accounts, and log in the app with their accounts. A user can send a friend request to another user, the receiver can decide whether to make this friend. Whenever a user performs any action, a time and location stamp will be stored. Each user can get a list of all their friends. Users can change their current state which will be useful when it comes to filters.

Users can post notes with some text, and set time and location limitations. Besides, users can also add tags to notes, which is a customized type to distinguish different varieties of notes. Time limitation means if the current time is within a note's certain schedule, this note is available. Location limitation means if someone is within the radius of the location of the note, he is able to receive this note. What's more, a user can decide whether others or only himself can receive the notes he posted. A note can be commented by every users.

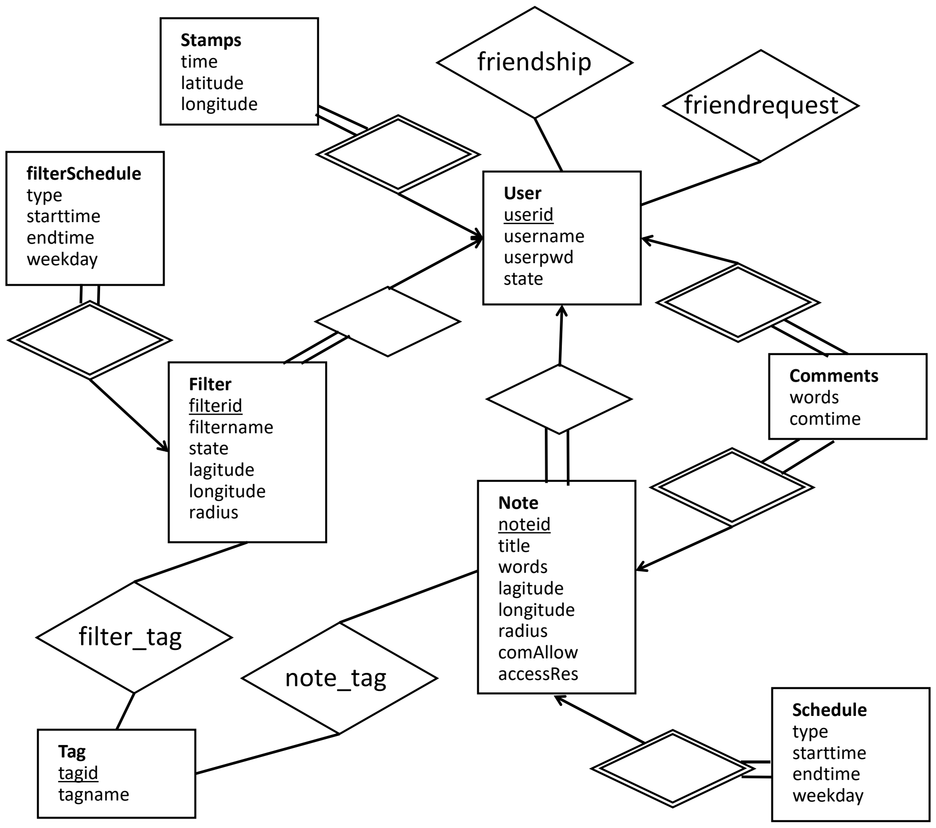
Users can set different filters for themselves. Filters can help users get the notes they want based on different conditions, such as time, location and tags. Every filter is bond with a state name. A filter works only if the user of this filter is in the same state. A user can get the notes which can satisfy at least one of his filters.

There's more explanations about schedule part. Oingo provides three different types of schedule. The first one is no time limitation which means a filter will get all the notes or a note will be received by all the filters. The second one is that there is a start time and a end time, filters and notes only work between that time period. The last one is based on weekdays. Users can set a time period for a certain weekday, filters and notes only work on that day between that period.

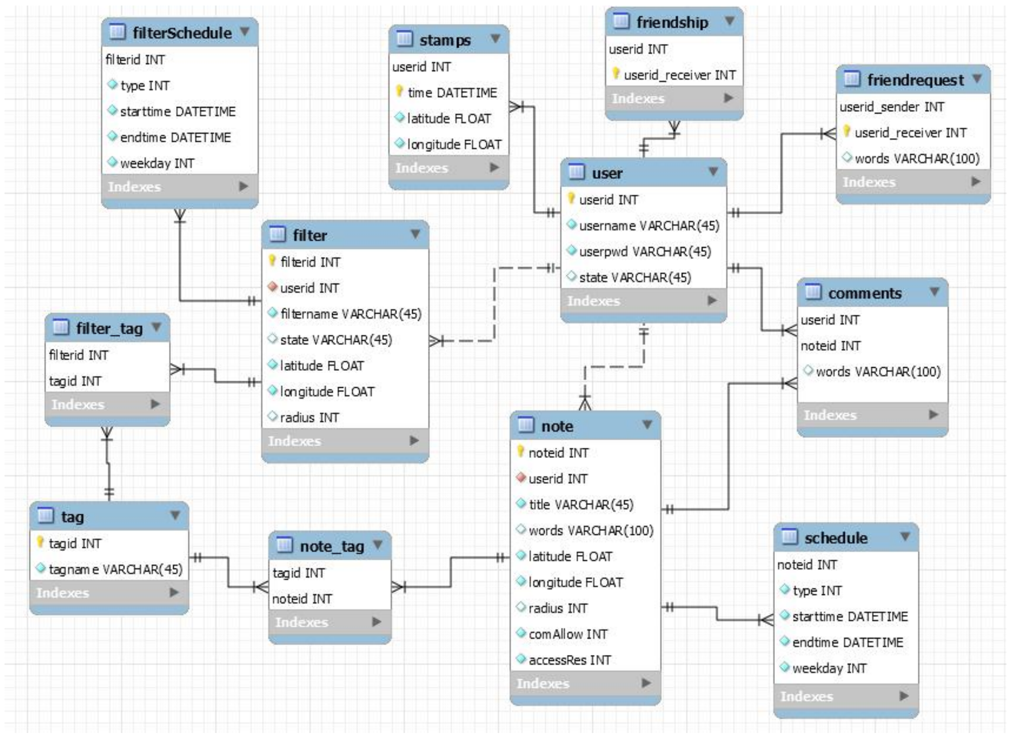
Last but not least, users can copy a url of a certain note or a user web page and share it with others. People can use that url to visit the pages even without login which is a very convenient function for app using.

2. Database Design

ER-diagram:



Relational format diagram:



Schema table:

user(userid, username, userpwd, state)
primary key:userid (auto increment)

friendship(userid, userid_receiver)
primary key:userid, userid_receiver
foreign key:userid referencing user.userid

note(noteid, userid, title, words, latitude, longitude, radius, comAllow, accessRes)
primary key:noteid (auto increment)
foreign key:userid referencing user.userid

schedule(noteid, type, starttime, endtime, weekday)
primary key:noteid
foreign key:noteid referencing note.noteid

tag(tagid, tagname)
primary key:tagid (auto increment)

note_tag(tagid, noteid)
primary key:tagid, noteid
foreign key:tagid referencing tag.tagid, noteid referencing note.noteid

comments(userid, noteid, words, comtime)
primary key:userid, noteid, comtime
foreign key:userid referencing user.userid, noteid referencing note.noteid

friendrequest(userid_sender, userid_receiver, words)
primary key:userid_sender, userid_receiver
foreign key:userid_sender referencing user.userid, userid_receiver referencing user.userid

filter(filterid, userid, filtername, state, latitude, longitude, radius)
primary key:filterid (auto increment)
foreign key:userid referencing user.userid

stamps(userid, time, latitude, longitude)
primary key:userid, time
foreign key:userid referencing user.userid

filter_tag(filterid, tagid)
primary key:filterid, tagid
foreign key:filterid referencing filter.filterid, tagid referencing tag.tagid

filterSchedule(filterid, type, starttime, endtime, weekday)
primary key:filterid
foreign key:filterid referencing filter.filterid

Assumptions:
There is at most one friend request between two certain users.
There is at most one friendship between two certain users.

There is at most one relation between a note and a tag.
There is at most one relation between a filter and a tag.

All comments follow a note, rather than a comment.
Each note and filter must have at least one tag.

3. Database Test

Sample data:
There are 3 users named 'Alice', 'Bob', 'Raven'. Alice and Bob are friends. Bob and Raven are friends. Raven sends a friend request to Alice.

notes
Alice writes a note about a pizza restaurant to recommend it
title:pizza text:good pizza restaurant!
tag:#food
time:2018-11-18 9:00->2018-12-30 9:00
location:(116.000000 , 39.000000)
comment:Bob makes a comment.

Bob writes a note about a gym to remind himself of practising
title:Great gym text:It’s cheap.
tag:#sports
time:Each Friday 17:00->19:00
location:(116.00001 , 38.99999)

Raven writes a note about a shop to remind buying Christmas gift.
title:shopping mall text:need to buy Christmas gift.
tag:#me, #shop
time:2018-12-01 9:00->2018-12-24 18:00
location:(40 , 74)

filters
Alice has a filter about food

title:EAT TIME
state: null
tag:#food
time:Each Sunday
location:(116 ,   39)

Bob has a filter about almost everything
title: a filter
state: null
tag:#food #sports #shop
time:2018-11-18 9:00->2018-12-30 9:00
location:(116,   39)

Raven has a filter about himself
title:a remind
state:leisure time
tag:#me
time:all the time
location:(40 ,   74)

Queries:
(1) Create a new user account, with name, login, and password.
sql query:
```
insert into user(username, userpwd) values ($name , $password);
```

(2) Add a new note to the system, together with tags, and spatial and temporal constraints.
sql query:
```
INSERT INTO `note` VALUES (null, $userid, $title, $text, $latitude,
$longitude, 1, 1, 1);
b = LAST_INSERT_ID();
INSERT INTO `schedule` VALUES (b, $scheduleType, $starttime, $endtime,
$weekday);
INSERT INTO `tag` VALUES (null, $tagname);
c = LAST_INSERT_ID();
INSERT INTO `note_tag` VALUES (c, b);
```

(3) For a given user, list all her friends.
sql query:
```
select u2.username
from friendship, user as u1, user as u2
where friendship.userid = u1.userid and friendship.userid_receiver =
u2.userid and u1.username = 'Bob';
```

(4) Given a user and her current location, current time, and current state, output all notes that she should currently be able to see given the filters she has set up.
sql query:

```
select distinct user.username,note.noteid, note.title, note.words,
note.latitude, note.longitude
       from (
           select distinct tag.tagid
           from filter, filterSchedule, user , filter_tag, tag
           where
               filter_tag.filterid=filter.filterid and
tag.tagid=filter_tag.tagid
               and filter.userid=user.userid and filter.userid=$userid
and filter.filterid=filterSchedule.filterid
               and (user.state=filter.state or filter.state = '')
               and (filterSchedule.type=1
               or (filterSchedule.type=2 and '$nowdate $nowtime' between
filterSchedule.starttime and filterSchedule.endtime)
               or (filterSchedule.type=3 and weekday('$nowdate
$nowtime')=filterSchedule.weekday and time('$nowdate $nowtime') between
time(filterSchedule.starttime) and time(filterSchedule.endtime))
               )
               and (filter.radius=0 or
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((filter.latitude*PI()/180-$latitud
e*PI()/180)/2),2)+COS(filter.latitude*PI()/180)*COS($latitude*PI()/18
0)*POW(SIN((filter.longitude*PI()/180-$longitude*PI()/180)/2),2)))*10
00)<filter.radius)
       )F, note, schedule ,user ,tag, note_tag
       where
           user.userid= note.userid
           and note.noteid=note_tag.noteid and tag.tagid=note_tag.tagid
and tag.tagid=F.tagid
           and note.noteid=schedule.noteid
           and (note.accessRes!=0 or note.userid=$userid)
           and (schedule.type=1
           or (schedule.type=2 and '$nowdate $nowtime' between
schedule.starttime and schedule.endtime)
           or (schedule.type=3 and weekday('$nowdate
$nowtime')=schedule.weekday and time('$nowdate $nowtime') between
time(schedule.starttime) and time(schedule.endtime))
           )
           and (note.radius=0 or
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((note.latitude*PI()/180-$latitude*
PI()/180)/2),2)+COS(note.latitude*PI()/180)*COS($latitude*PI()/180)*P
OW(SIN((note.longitude*PI()/180-$longitude*PI()/180)/2),2)))*1000)<no
te.radius);
```

(5) Given a note (that maybe was just added to the system) and the current time, output all users that should currently be able to see this note based on their filter and their last recorded location.
sql query:

```
with F as(select filter.filterid, filter.userid, filter.state,
filter.latitude, filter.longitude, filter.radius
from filter, filterSchedule
where filter.filterid=filterSchedule.filterid and
(filterSchedule.type=1
or (filterSchedule.type=2 and $currenttime between
filterSchedule.starttime and filterSchedule.endtime)
or (filterSchedule.type=3 and
weekday($currenttime)=filterSchedule.weekday and time($currenttime)
between time(filterSchedule.starttime) and
time(filterSchedule.endtime))
))
select distinct user.username
from F, note, filter_tag, note_tag, user, (
select *
from(select *
from stamps
order by time DESC
limit 999999)as t
group by t.userid
)as st
where user.userid=F.userid and (user.state is null or
user.state=F.state)and F.userid=st.userid and note.noteid=1 and
F.filterid=filter_tag.filterid and note.noteid=note_tag.noteid and
filter_tag.tagid=note_tag.tagid
and
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((F.latitude*PI()/180-st.latitude*P
I()/180)/2),2)+COS(F.latitude*PI()/180)*COS(st.latitude*PI()/180)*POW
(SIN((F.longitude*PI()/180-st.longitude*PI()/180)/2),2)))*1000)<F.rad
ius and
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((note.latitude*PI()/180-st.latitud
e*PI()/180)/2),2)+COS(note.latitude*PI()/180)*COS(st.latitude*PI()/18
0)*POW(SIN((note.longitude*PI()/180-st.longitude*PI()/180)/2),2)))*10
00)<note.radius );
```

(6) In some scenarios, in very dense areas or when the user has defined very general filters, there may be a lot of notes that match the current filters for a user. Write a query showing how the user can further filter these notes by inputting one or more keywords that are matched against the text in the notes using the contains operator.
sql query:

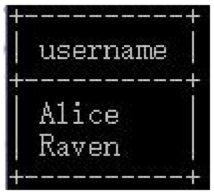
```
select distinct user.username,note.noteid, note.title, note.words,
note.latitude, note.longitude
        from (
            select distinct tag.tagid
            from filter, filterSchedule, user , filter_tag, tag
            where
                filter_tag.filterid=filter.filterid and
tag.tagid=filter_tag.tagid
                and filter.userid=user.userid and filter.userid=$userid
and filter.filterid=filterSchedule.filterid
                and (user.state=filter.state or filter.state = '')
                and (filterSchedule.type=1
                or (filterSchedule.type=2 and '$nowdate $nowtime' between
filterSchedule.starttime and filterSchedule.endtime)
                or (filterSchedule.type=3 and weekday('$nowdate
$nowtime')=filterSchedule.weekday and time('$nowdate $nowtime') between
time(filterSchedule.starttime) and time(filterSchedule.endtime))
                )
                and (filter.radius=0 or
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((filter.latitude*PI()/180-$latitud
e*PI()/180)/2),2)+COS(filter.latitude*PI()/180)*COS($latitude*PI()/18
0)*POW(SIN((filter.longitude*PI()/180-$longitude*PI()/180)/2),2)))*10
00)<filter.radius)
        )F, note, schedule ,user ,tag, note_tag
        where
            user.userid= note.userid
            and note.noteid=note_tag.noteid and tag.tagid=note_tag.tagid
and tag.tagid=F.tagid
            and note.noteid=schedule.noteid and (note.title REGEXP
concat_ws('|' ,"buy" ,"pizza") or note.words REGEXP
concat_ws('|' ,"buy" ,"pizza"))
            and (note.accessRes!=0 or note.userid=$userid)
            and (schedule.type=1
            or (schedule.type=2 and '$nowdate $nowtime' between
schedule.starttime and schedule.endtime)
            or (schedule.type=3 and weekday('$nowdate
$nowtime')=schedule.weekday and time('$nowdate $nowtime') between
time(schedule.starttime) and time(schedule.endtime))
            )
            and (note.radius=0 or
ROUND(6378.138*2*ASIN(SQRT(POW(SIN((note.latitude*PI()/180-$latitude*
PI()/180)/2),2)+COS(note.latitude*PI()/180)*COS($latitude*PI()/180)*P
OW(SIN((note.longitude*PI()/180-$longitude*PI()/180)/2),2)))*1000)<no
te.radius) ;
```

Test result:

(1) Insertion succeed.

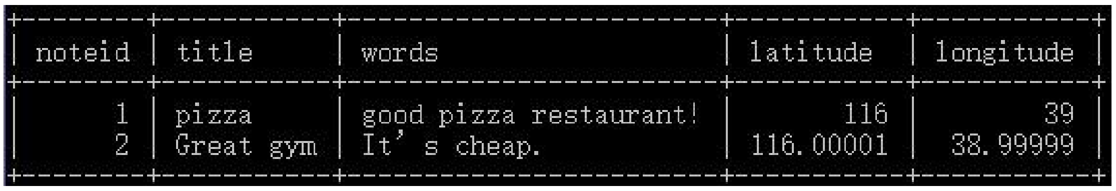
(2) Insertions succeed.

(3) list all Bob's friends.



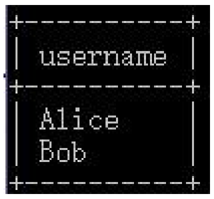
```
+----------+
| username |
+----------+
| Alice    |
| Raven    |
+----------+
```

(4) Output all the notes Bob is able to see, current time is '2018-12-14 18:00:00'



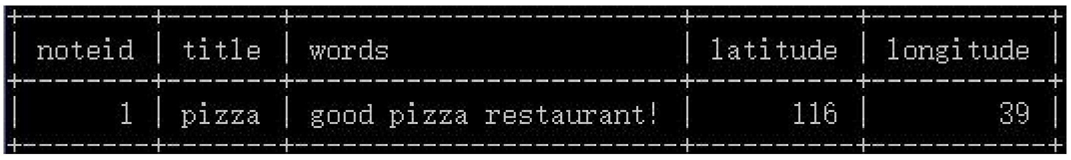
```
+--------+-----------+------------------------+-----------+-----------+
| noteid | title     | words                  | latitude  | longitude |
+--------+-----------+------------------------+-----------+-----------+
|      1 | pizza     | good pizza restaurant! |       116 |        39 |
|      2 | Great gym | It’ s cheap.           | 116.00001 |  38.99999 |
+--------+-----------+------------------------+-----------+-----------+
```

(5) Choose the node written by Alice, current time is '2018-12-16 15:00:00'



```
+----------+
| username |
+----------+
| Alice    |
| Bob      |
+----------+
```

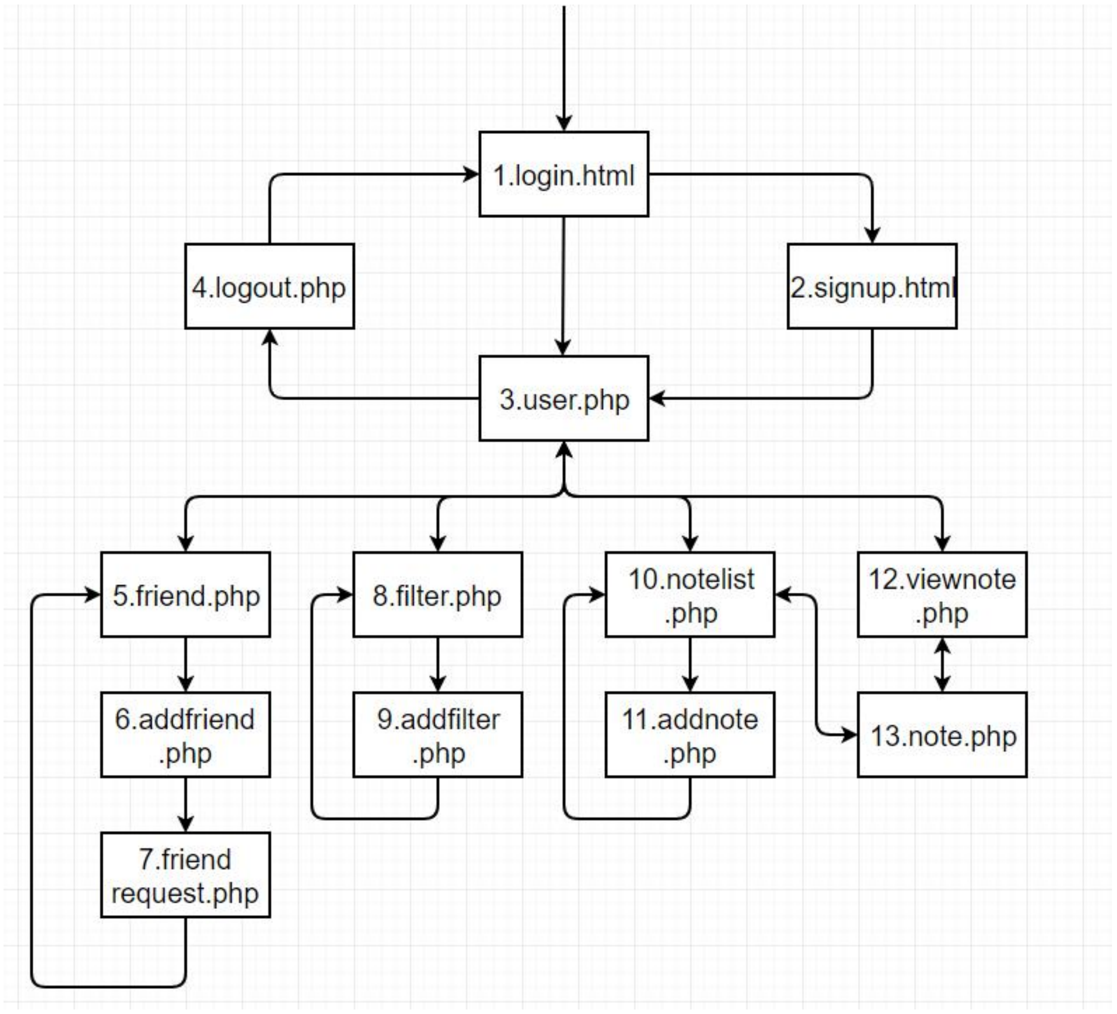
(6) based on(4), add keywords like “buy” “pizza”



```
+--------+-------+------------------------+----------+-----------+
| noteid | title | words                  | latitude | longitude |
+--------+-------+------------------------+----------+-----------+
|      1 | pizza | good pizza restaurant! |      116 |        39 |
+--------+-------+------------------------+----------+-----------+
```

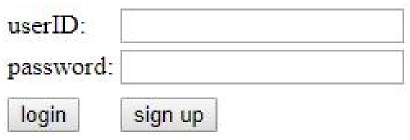
4. How to use this app

Diagram of web pages:



Each web page:

(1) log in page



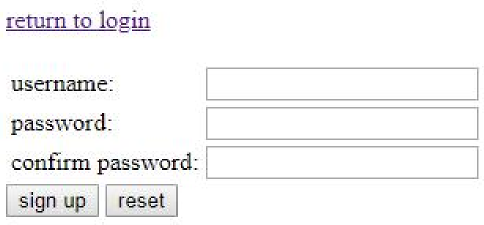
A simple login interface.

If you input a wrong user id or a wrong password, you will not manage to log in.

If you click the 'sign up' button, you will skip to sign up page.

If you succeed to log in, you will skip to user center and get access to main functions of the app.

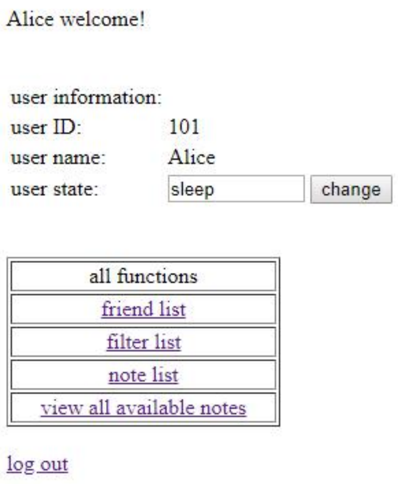
(2) sign up page


return to login

username:	
password:	
confirm password:	

A simple sign up interface.
In here, you need to input your user name and password. Make sure the confirm password is same as your password. After signing up, you will get your unique user ID, do not forget that!

(3) user center page


Alice welcome!

user information:

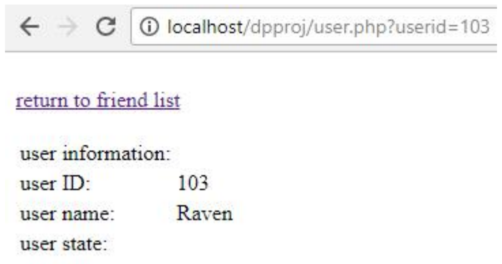
user ID:	101
user name:	Alice
user state:	sleep

all functions
friend list
filter list
note list
view all available notes

log out

This is the main page after you log in or sign up.
In here, you can change your state or visit several functions by clicking the hyper links.

Besides, you can visit others' user page through URLs, like the following


localhost/dpproj/user.php?userid=103

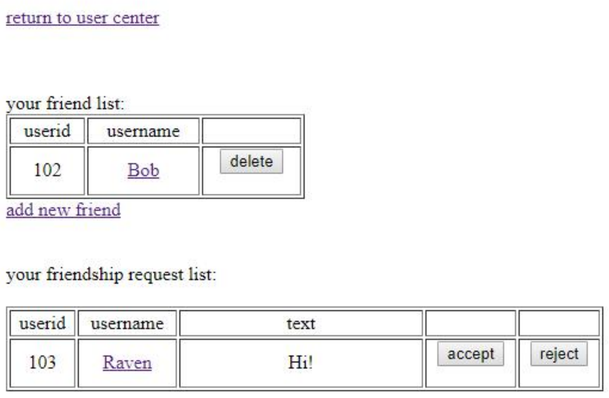
return to friend list

user information:

user ID:	103
user name:	Raven
user state:	

(4) log out action
After clicking 'log out', you run back to login page with the cookie refreshed.

(5) friend list page


return to user center

your friend list:

userid	username	
102	Bob	delete

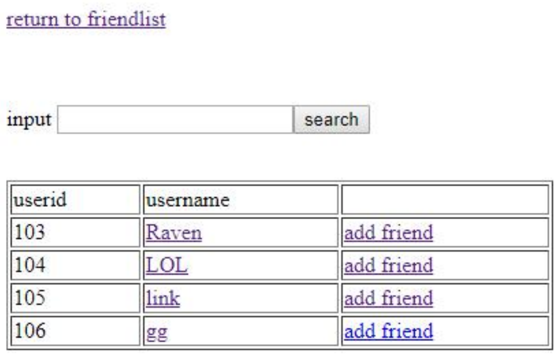
add new friend

your friendship request list:

userid	username	text		
103	Raven	Hi!	accept	reject

On this page, you can get your friend list and your friendship request list.
You can click 'delete' button near one of your friends to end the relationship.
You can click 'accept' or 'reject' button near one of your requests to handle this.
You can click the user's name to get into their user pages.
You can click 'add new friend' link to visit the add friend page.

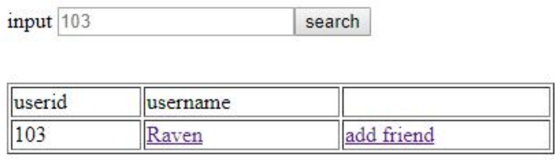
(6) add friend page


return to friendlist

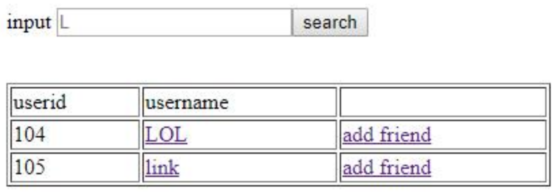
input search

userid	username	
103	Raven	add friend
104	LOL	add friend
105	link	add friend
106	gg	add friend

On this page, you can get a list of all potential users who would be your friends.
You can also find a user by a specific id or a text piece.


input 103 search

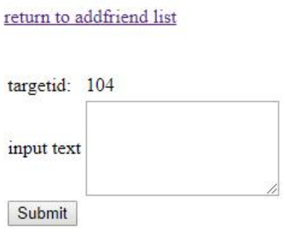
userid	username	
103	Raven	add friend


input L search

userid	username	
104	LOL	add friend
105	link	add friend

When you click 'add friend' link, you will visit the friend request page.

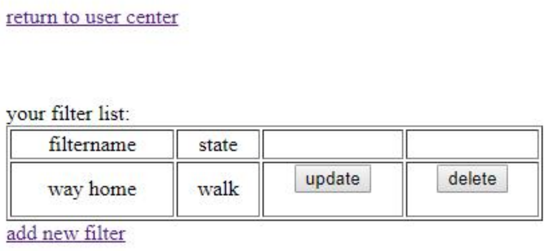
(7) friend request page


return to addfriend list

targetid:	104
input text	

You can use this simple page to send a friend request to a user. At the same time, you can have only one friend request to a certain user.

(8) filter list page


return to user center

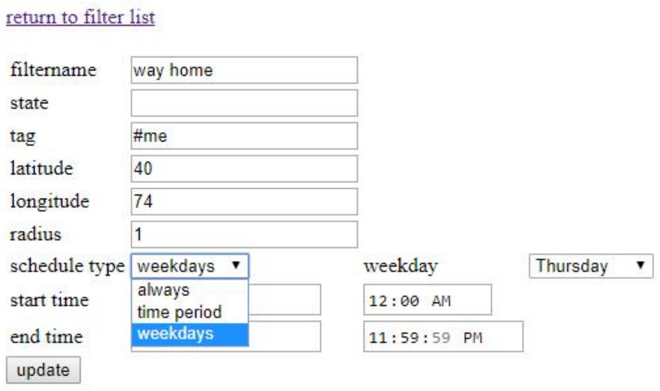
your filter list:

filtername	state		
way home	walk	update	delete

add new filter

On this page, you can get your filter list.
You can click 'update' or 'delete' button near one of your filters to make some change to it.
You can click 'add new filter' link to visit the add filter page.

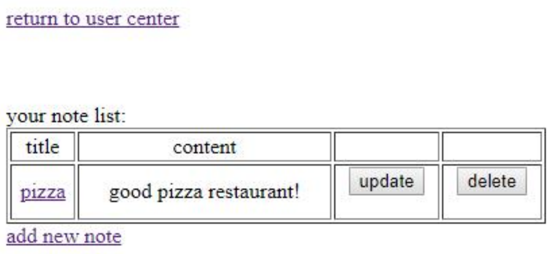
(9) add filter page


return to filter list

filtername	way home		
state			
tag	#me		
latitude	40		
longitude	74		
radius	1		
schedule type	weekdays	weekday	Thursday
start time			12:00 AM
end time			11:59:59 PM

You can use this page to add or update a filter.
On this page, you need to fill in a filter form. State means this filter will work only if your user state is the same. You need to add at least one tag to the filter, otherwise no notes will match this filter. Three schedule types work differently so that you need to set your schedule carefully.

(10) note list page


return to user center

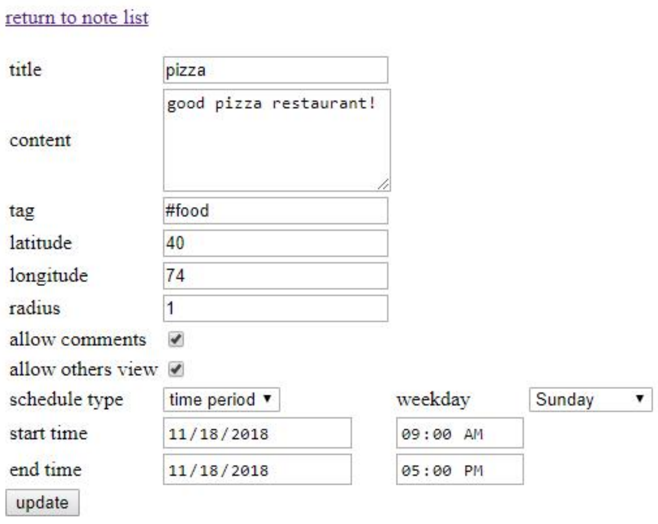
your note list:

title	content		
pizza	good pizza restaurant!	update	delete

add new note

On this page, you can get note list of yourself.
You can click 'update' or 'delete' button near one of your notes to make some change to it.
You can click 'add new note' link to visit the add note page.

(11) add note page

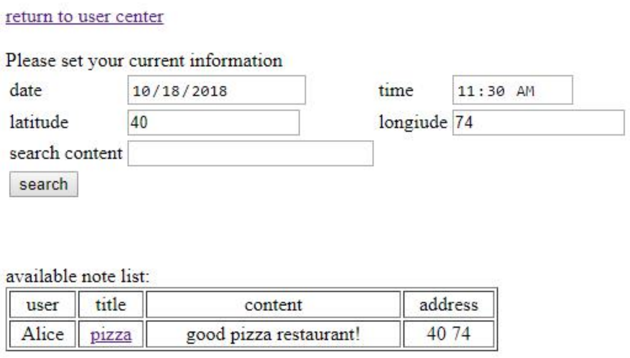

return to note list

title	pizza		
content	good pizza restaurant!		
tag	#food		
latitude	40		
longitude	74		
radius	1		
allow comments	✓		
allow others view	✓		
schedule type	time period	weekday	Sunday
start time	11/18/2018	09:00 AM	
end time	11/18/2018	05:00 PM	

update

You can use this page to add or update a note.
On this page, you need to fill in a note form. You need to add at least one tag to the note, otherwise no filters will match this note.
You can choose whether this note allows comments.
You can choose whether this note allows others' viewing.

(12) view note page


return to user center

Please set your current information

date	10/18/2018	time	11:30 AM
latitude	40	longiude	74
search content			

search

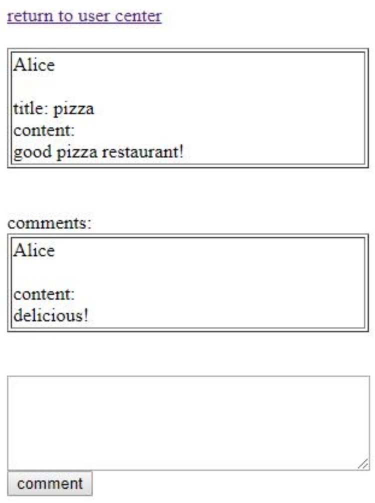
available note list:

user	title	content	address
Alice	pizza	good pizza restaurant!	40 74

For convenience, you can input your current stamp to simulate a real app condition.

After clicking 'search' button, you will get all the available notes you can get through your filters.
You can choose to only get the notes that contains your search terms in title or content.
You can click note title link to get into a note page.

(13) note page


return to user center

Alice

title: pizza
content:
good pizza restaurant!

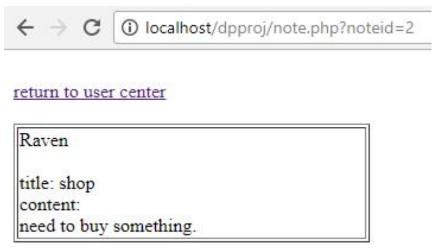
comments:

Alice

content:
delicious!

In here, you can get the content of the specific note and comments below it.
After login, you can also make comments by inputting some sentence and clicking 'comment' button.

Besides, you can visit a note page through URLs, like the following


localhost/dpproj/note.php?noteid=2

return to user center

Raven

title: shop
content:
need to buy something.

5. Project Design

Functions of each files:
(1) 'login.html'
To run this project properly, you need to visit this page first.
It is basically a html file providing the interface.

(2) 'signup.html'
It is visited after user clicking 'sign up' button on 'login.html'.
It is basically a html file providing the interface.

(3) 'signup.php'
It is visited after user clicking 'sign up' button on 'signup.html'.
It checks whether the two passwords are same and then create a new user account through SQL insert codes. The user ID of this account will be shown on the page.

(4) 'user.php'
It is visited after user clicking 'log in' on 'login.html' or trying to view a certain user's page by adding '?userid=X' to URL.
It can get the _POST information from 'login.html'. It first uses SQL query to check whether the user ID exists, and then checks whether the password matches the user ID. After this successful first log in, it writes user information into cookie.
What's more, it can get the _POST information from itself which contains a state text. By this, it performs a SQL query to update user's state.
It can get the _GET information from URL if there is a userid attached to it. Page content will be a bit different based on whether the viewer is a visitor or a user. If a viewer happens to be the owner of this page, then several functions will be shown on this page.

(5) 'logout.php'
It is visited after user clicking 'log out' link on 'user.php'.
It cleans the cookie information so that user needs to log in before visiting some pages.

(6) 'friend.php'
It is visited after user clicking 'friend list' link on 'user.php'.
It contains both friend list interface and friend request interface code.
It can get the _POST information from itself which contains action type and target information. It performs a SQL query if the action is deleting a friend or handling a friend request.

(7) 'addfriend.php'
It is visited after user clicking 'add new friend' link on 'friend.php'.
It visits database to get a table of users and show them on the page.
It can get the _POST information from itself which contains a search text. With this text, it uses 'like' in SQL query to get search targets.
It can also get the the _POST information from 'friendrequest.php' which contains some necessary information of a friend request. Then it performs a SQL insert query.

(8) 'friendrequest.php'
It is visited after user clicking one of the 'add friend' links on 'addfriend.php'.
It can get the _POST information from 'addfriend.php' which contains a target user id. It first checks if there already exists a friend request from the user to the target. If not, it provides a interface for user to type in a simple friend request.

(9) 'filter.php'
It is visited after user clicking 'filter list' link on 'user.php'.
It provides a friend list interface.

It can get the _POST information from both itself and 'addfilter.php' which contains action type and target information.
If it receives a delete action, it will use SQL query to delete all the tag and schedule relations of the filter and finally delete the filter.
If it receives a create action, it will use SQL query to insert a new filter and then split the tag text into several tags. It inserts all the tags not existing yet and insert all the tag and schedule relations of the filter.
If it receives a update action, it will combine the delete and create action to achieve the same effect.

(10) 'addfilter.php'
It is visited after user clicking 'add new filter' link or 'update' button on 'filter.php'.
It provides a filter form interface. If the action is from 'add new filter', the form is mostly blank by default. If the action is from 'update' button, it will read information from database first and fill the form with the information.

(11) 'notelist.php'
It is visited after user clicking 'note list' link on 'user.php'.
Its function is similar to 'filter.php'. It provides a note list interface and helps handle different note actions from itself and 'addnote.php'.

(12) 'addnote.php'
It is visited after user clicking 'add new note' link or 'update' button on 'notelist.php'.
Its function is similar to 'addfilter.php'. It provides a note form interface and initializes the form.

(13) 'viewnote.php'
It is visited after user clicking 'view all available notes' link on 'user.php'.
It provides some input texts and a note list interface.
It can get the _POST information from itself which contains current stamp and search text. It uses SQL query to get filters and notes both fitting the condition. If there is at least one filter left, it means the user is open to this current condition, and all the notes it gets will be shown in the note list.

(14) 'note.php'
It is visited after user clicking a note’s title or trying to view a certain note page by adding '?noteid=X' to URL.
It uses SQL query to get the information of this note and all the related comments, then it shows the content of the note, the comments below and a comment input text for user.
It can get the _POST information from itself which contains a comment text. If it confirms that the viewer is not a visitor, it inserts this comment.

(15) 'conn.php'
This is a php file included by others. It provides basic database connection codes and a function trySQLs() to handle transactions.

Safety:

All the input values are applied with mysqli_real_escape_string() function in order to get rid of situations that hackers can change SQL function by some tricky text. Besides, all the output values are applied with htmlspecialchars() function in order to prevent hackers from changing html behavior by some tricky text.

Transactions in concurrency:

When it comes to a atomic sequence of queries containing functions like 'INSERT', 'UPDATE' and 'DELETE'. This sequence will be put into a transaction by using mysqli_begin_transaction(), mysqli_commit(). 'ROLLBACK' SQL query is used to return to the start save point if failure happens. To make sure transaction function is available, you should open the 'my.ini' file of MYSQL and change 'default-storage-engine=MYISAM' to 'default-storage-engine=INNODB' because MYISAM does not support transaction function. After that you can create the whole database.