CS6083 Project Oingo

Ke Fan | N19676142 | kf1550@nyu.edu

Jiexin Xu | N11809058 | jx1119@nyu.edu

Introduction

`Oingo` is an application that allows users to share useful information via their mobile devices based on social, geographic, temporal, and keyword constraints.

Database Management System

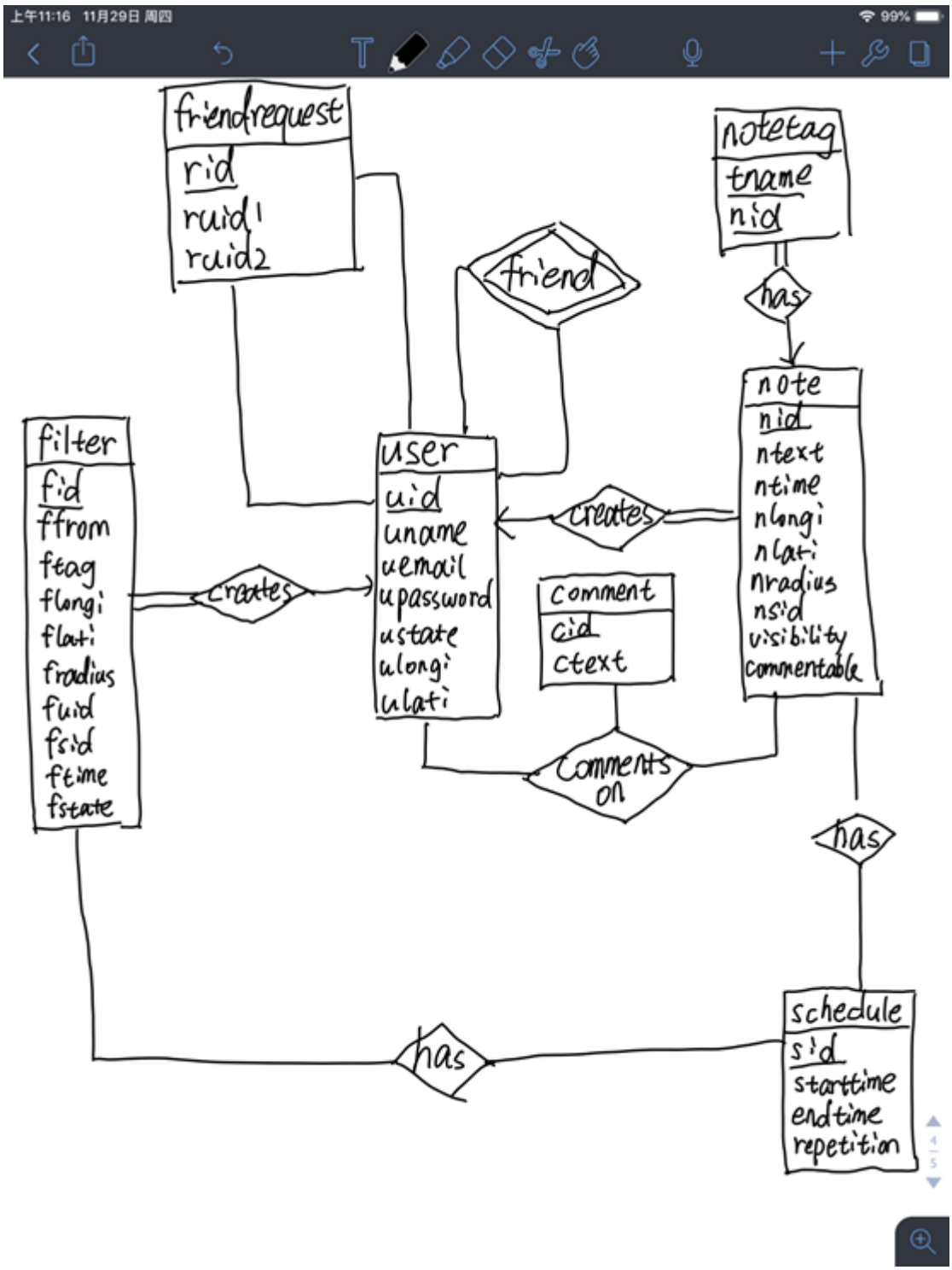
Relational Schema

```
user(uid,uname,uemail,upassword,ustate,ulongi,ulati)
primary key: uid

friend(uid1,uid2)
primary key: uid1, uid2
friend.uid1,uid2 is a foreign key referencing user.uid

friendRequest(rid,ruid1,ruid2)
primary key: rid
friendRequest.ruid1,ruid2 is a foreign key referencing user.uid

note(nid,ntext, ntime,nlongi,nlati,nradius,nsid,createdBy,visibility,commentable)
primary key: nid
note.createdBy is a foreign key referencing user.uid
note.nsid is a foreign key referencing schedule.sid

notetag(tname,nid)
primary key: tname, nid
notetag.nid is a foreign key referencing note.nid

schedule(sid,starttime,endtime,repetition)
primary key: sid

comment(cid,nid,ctext,createdBy)
primary key: cid
comment.nid is a foreign key referencing note.nid
comment.createdBy is a foreign key referencing user.uid


filter(fid,ffrom,ftag,flongi,flati,fradius,fuid,fstate)
primary key: fid
filter.fuid is a foreign key referencing user.uid
```

Views

```
activedNotes(uid,nid,nBelong)
applyFilter(uid,nid)
activedFilter(fid,ffrom,ftag,fuid,fstate)
```

E-R Model



Explanations of Design

1. User

In this table,

`uid` is a unique id of each user, it is used to identify different users.

`uname` is the user name; `uemail` is the email address of a user.

`uname` and `uemail` are setup when user sign up for the first time.

`upassword`, of course, is the password for this account.

`ustate` is the current state of a user, every user can edit their current state if they want.

`ulongi` and `ulati` are the keys representing longitude and latitude, by these two keys, a user's location can be determined, and distance can be calculated if we know the longitude and latitude of two places.

2. Friend

Table `friend` consists of friendships of users, two users are friends if their `uid` is in the same row.

Like if user 1 and user 2 are friends, there exists (1,2) in table friend, friendship is mutual.

3. FriendRequest

`FriendRequest` is a request that a user send to another in order to build up friendship, say user 1 make a request to user 2 to be friends, this request will be written into the database, and user 2 will receive the request from server, if user 1 response as consent, their friendship will be put into table friend, thus they become friends.

4. Note

In table `Note`, each note is identified by an `nid`.

`ntext` refers to the text in this note.

`ntime` is the time when the note was created.

`nlongi` and `nlati` refers to longitude and latitude.

`nradius` is the radius available for the note, which is setup by the user who published this note.

`nsid` is the id of schedule related to this note.

`visibility` can have 3 values : `self`, `friend` and `every`, this is used to determine which user can see this note.

And for `commentable`, it has two values, yes means people can leave comment for this note, vise versa.

5. Notetag

`Notetag` is used to describe tags that a note has, since a note can have several tags, so both `tname` and `nid` are primary keys.

6. Schedule

`Schedule` table consists of schedules for notes that can be seen. `sid` is a key to identify different schedules. And for `repetition`, we use integers to represent days, for example, if we have value 1 for repetition, that means this schedule will perform once a day. Other frequencies such as a week can be represented as 7.

We know this is not a perfect way for solution, but we will modify it in the second step of this project.

7. Comment

`Comment` is created by a user, before we insert a comment into the database, we will check the `nid` in table note to see if this user has permission to make a comment to this note, if he has, his comment will be recorded in `ctext`.

8. Filter

A user can have different filters, on what they want to see based on their current location, the current time, and maybe also their current state (such as “at work”, “lunch break”, “just chilling”, or whatever they choose as the description).

9. Views

In this schema, we use 3 auxiliary views to help us get the work done.

View `activedNotes` is all `(uid, nid)` groups filtered based on the current time, the current location of each user, and the publisher's permission settings for the note. View `activedNotes` is all `(fid, ffrom, ftag, fuid, fstate)` groups filtered based on the current time, the current location of each user, and the user's settings for the filter.

View `applyFilter` return the result based on `activedNotes` and `activedFilter`.

Build Database

```
CREATE TABLE `user` (
`uid` int(11) NOT NULL,
`uname` varchar(255) NOT NULL,
`uemail` varchar(255) NOT NULL,
`upassword` varchar(16) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`ustate` varchar(50) NULL,
`ulongi` varchar(255) NULL,
`ulati` varchar(255) NULL,
PRIMARY KEY (`uid`)
);

CREATE TABLE `note` (
`nid` int(11) NOT NULL,
`ntext` varchar(255) NOT NULL,
`ntag` varchar(20) NULL,
`ntime` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP,
`nlongi` varchar(255) NOT NULL,
`nlati` varchar(255) NOT NULL,
`nradius` int(11) NOT NULL,
`nsid` int(11) NULL,
`createdBy` int(11) NOT NULL,
```

```
`visibility` varchar(255) NOT NULL,
`commentable` varchar(10) NULL,
PRIMARY KEY (`nid`)
);

CREATE TABLE `friend` (
`uid1` int(11) NOT NULL,
`uid2` int(11) NOT NULL,
PRIMARY KEY (`uid1`, `uid2`)
);

CREATE TABLE `schedule` (
`sid` int(11) NOT NULL,
`starttime` datetime NULL ON UPDATE CURRENT_TIMESTAMP,
`endtime` datetime NULL ON UPDATE CURRENT_TIMESTAMP,
`repetition` int(11) NULL,
PRIMARY KEY (`sid`)
);

CREATE TABLE `filter` (
`fid` int(11) NOT NULL,
`ffrom` varchar(11) NULL,
`ftag` varchar(20) NULL,
`flongi` varchar(255) NULL,
`flati` varchar(255) NULL,
`fradius` int(11) NULL,
`fuid` int(11) NULL,
`fsid` int(11) NULL,
`ftime` datetime NULL ON UPDATE CURRENT_TIMESTAMP,
`fstate` varchar(50) NULL,
PRIMARY KEY (`fid`)
);

CREATE TABLE `friendRequest` (
`rid` int(11) NOT NULL,
`ruid1` int(11) NOT NULL,
`ruid2` int(11) NOT NULL,
PRIMARY KEY (`rid`)
);

CREATE TABLE `noteTag` (
`tname` varchar(20) NOT NULL,
`nid` int(11) NOT NULL,
PRIMARY KEY (`tname`, `nid`)
);

CREATE TABLE `comment` (
`cid` int(11) NOT NULL AUTO_INCREMENT,
`nid` int(11) NOT NULL,
`ctext` varchar(255) NOT NULL,
`createdBy` int(11) NOT NULL,
PRIMARY KEY (`cid`)
);
```

```
ALTER TABLE `friend` ADD CONSTRAINT `uid1_fk` FOREIGN KEY (`uid1`) REFERENCES `user`
(`uid`);
ALTER TABLE `friend` ADD CONSTRAINT `uid2_fk` FOREIGN KEY (`uid2`) REFERENCES `user`
(`uid`);
ALTER TABLE `note` ADD CONSTRAINT `sid_fk` FOREIGN KEY (`nsid`) REFERENCES `schedule`
(`sid`);
ALTER TABLE `filter` ADD CONSTRAINT `fuid_fk` FOREIGN KEY (`fuid`) REFERENCES `user`
(`uid`);
ALTER TABLE `note` ADD CONSTRAINT `cretedBy` FOREIGN KEY (`createdBy`) REFERENCES
`user` (`uid`);
ALTER TABLE `friendRequest` ADD CONSTRAINT `ruid_fk` FOREIGN KEY (`ruid1`) REFERENCES
`user` (`uid`);
ALTER TABLE `noteTag` ADD FOREIGN KEY (`nid`) REFERENCES `note` (`nid`);
ALTER TABLE `comment` ADD FOREIGN KEY (`nid`) REFERENCES `note` (`nid`);
ALTER TABLE `comment` ADD FOREIGN KEY (`createdBy`) REFERENCES `user` (`uid`);
```

```
CREATE
VIEW `actived Notes` AS
SELECT uid, nid, createdBy as nBelong
FROM
    `user`
    NATURAL JOIN note
    NATURAL JOIN `schedule`
WHERE ('2018-11-27 20:51:38' BETWEEN starttime AND endtime)
AND DATEDIFF('2018-11-27 20:51:38', ntime) % `schedule`.repetition = 0
AND ABS(SQRT(POW(`user`.ulongi,2) + POW(`user`.ulati,2)) - SQRT(POW(nlongi,2) +
POW(nlati,2))) <= nradius
AND note.visibility = 'all' OR (note.visibility = 'friend ' AND uid in (SELECT uid2
FROM friend WHERE uid1 = note.createdBy UNION ALL SELECT uid1 from friend WHERE uid2 =
note.createdBy)) OR(note.visibility = 'self' and note.createdBy = uid);
```

```
CREATE
VIEW `actived Filter` AS
SELECT DISTINCT fid, ffrom, ftag, fuid, fstate
FROM
    `user`
    JOIN filter f on `user`.uid = f.fuid
    JOIN `schedule` s2 ON s2.sid = f.fsid
WHERE ('2018-11-27 20:51:38' BETWEEN s2.starttime AND s2.endtime)
AND DATEDIFF('2018-11-27 20:51:38', ftime) % s2.repetition = 0
AND ABS(SQRT(POW(f.flongi,2) + POW(f.flati,2)) - SQRT(POW(`user`.ulongi,2) +
POW(`user`.ulati,2))) <= fradius;
```

```
CREATE
VIEW `applyFilter` AS
SELECT an.uid, an.nid
FROM actived Notes as an
JOIN actived Filter as f on an.uid = f.fuid
JOIN note as n ON an.nid = n.nid and f.ftag IN (SELECT tname FROM notetag WHERE an.nid
= notetag.nid)
JOIN `user` u ON an.uid = u.uid
WHERE f.fstate = u.ustate and  f.ffrom = 'all' OR (f.ffrom = 'friend' AND an.nBelong in
(SELECT uid2 FROM friend
WHERE uid1 = an.uid
UNION ALL
SELECT uid1 from friend
WHERE uid2 = an.uid)) OR (f.ffrom = 'me' and an.uid = an.nBelong);
```

Populate Database with Sample Data

```
# Record of User
INSERT INTO `user` VALUES (1, 'u1', 'u1@nyu.edu', '111', 'at work', '32.534167',
'66.078056 ');
INSERT INTO `user` VALUES (2, 'u2', 'u2@nyu.edu', '222', 'lunch', '36.948889',
'66.328611');
INSERT INTO `user` VALUES (3, 'u3', 'u3@nyu.edu', '333', 'drunk', '36.083056',
'69.0525');
```

uid	uname	uemail	upassword	ustate	ulongi	ulati
1	u1	u1@nyu.edu	111	state1	1	1
2	u2	u2@nyu.edu	222	state2	2	2
3	u3	u3@nyu.edu	333	state3	3	3

```
INSERT INTO `friend` VALUES (1, 2);
INSERT INTO `friend` VALUES (1, 3);
```

uid1	uid2
1	2
1	3

```
INSERT INTO `friendrequest` VALUES (1, 1, 3);
INSERT INTO `friendrequest` VALUES (2, 1, 2);
INSERT INTO `friendrequest` VALUES (3, 1, 2);
INSERT INTO `friendrequest` VALUES (4, 1, 2);
INSERT INTO `friendrequest` VALUES (5, 2, 3);
```

rid	ruid1	ruid2
1	1	3
2	1	2
3	1	2
4	1	2
5	2	3

```
INSERT INTO `note` VALUES (1, 'note1', '2018-11-29 09:47:25', '32.534167', '66.078056
', 100, 1, 1, 'all', 'yes');
INSERT INTO `note` VALUES (2, 'note2', '2018-11-29 09:47:30', '36.948889', '66.328611',
200, 2, 2, 'friends', 'yes');
INSERT INTO `note` VALUES (3, 'note3', '2018-11-29 09:42:03', '36.083056', '69.0525',
300, 3, 3, 'self', 'no');
INSERT INTO `note` VALUES (4, 'text4', '2018-11-29 13:33:46', '36.948889', '66.328611',
200, 4, 3, 'all', 'no');
```

nid	ntext	ntime	nlongi	nlati	nradius	nsid	createdBy	visibility	commentable
1	note1	2018-11-29 09:47:25	32.534167	66.078056	100	1	1	all	yes
2	note2	2018-11-29 09:47:30	36.948889	66.328611	200	2	2	friends	yes
3	note3	2018-11-29 09:42:03	36.083056	69.0525	300	3	3	self	no

```
INSERT INTO `notetag` VALUES ('tname1', 1);
INSERT INTO `notetag` VALUES ('tname3', 1);
INSERT INTO `notetag` VALUES ('tname5', 1);
INSERT INTO `notetag` VALUES ('tname2', 2);
INSERT INTO `notetag` VALUES ('tname4', 3);
```

tname	nid
tname1	1
tname2	2
tname3	1
tname4	3

```
INSERT INTO `schedule` VALUES (1, '2018-10-3 11:19:42', '2018-11-29 11:19:49', 0);
INSERT INTO `schedule` VALUES (2, '2018-11-26 20:12:26', '2018-12-5 20:12:26', 1);
INSERT INTO `schedule` VALUES (3, '2017-1-1 09:42:46', '2018-11-29 09:42:51', 7);
INSERT INTO `schedule` VALUES (4, '2018-11-21 00:00:00', '2018-12-31 11:59:59', 1);
```

sid	starttime	endtime	repetition
1	2018-10-03 11:19:42	2018-11-29 11:19:49	0
2	2018-11-26 20:12:26	2018-12-05 20:12:26	1
3	2017-01-01 09:42:46	2018-11-29 09:42:51	7

```
INSERT INTO `comment` VALUES (1, 1, 'comment1', 1);
INSERT INTO `comment` VALUES (2, 1, 'comment2', 2);
INSERT INTO `comment` VALUES (3, 2, 'comment3', 1);
INSERT INTO `comment` VALUES (4, 1, 'comment4', 3);
INSERT INTO `comment` VALUES (5, 2, 'comment5', 2);
```

cid	nid	ctext	createdBy
1	1	comment1	1
2	1	comment2	2
3	2	comment3	1
4	1	comment4	3
5	2	comment5	2

```
INSERT INTO `filter` VALUES (1, 'friend', '1', '32.534167', '66.078056 ', 300, 1, 1,
'2018-11-29 11:26:42', 'at work');
INSERT INTO `filter` VALUES (2, 'friend', '1', '32.534167', '66.078056 ', 300, 1, 1,
'2018-11-29 11:43:37', 'drunk');
```

fid	ffrom	ftag	flongi	flati	fradius	fuid	fsid	ftime	fstate
1	friend	1	32.534167	66.078056	300	1	1	2018-11-29 11:26:42	at work
2	friend	1	32.534167	66.078056	300	1	1	2018-11-29 11:43:37	drunk

Business Procedures

1. Register

A user can sign up for the first time, when registering, a user must provide a username, email address and password. Front end will send encrypted password to the server, then server will write these data into the database.

2. Updating states

When the user login for the first time, they are required to set up their status, this information will be sent together with the user's longitude and latitude.

3. Sending a friend request and making friends

A user can send a friend request to a specific user, the keyword for making this request can be another user's email address or username, server record this into database, and select the uid from user table, then server will send a request to the user been requested, if he permits, they become friend, thus make a new record into table friend.

4. Post a new note

When a user want to post a new note, the following fields cannot be null:

text, radius, visibility, whether can be comment. Longitude and latitude is automatically generated from the user's device. These information will be written into table note. A user can also set tags for a note, tags will be recorded in table note tag.

5. Make a schedule

A schedule can be used by several tags, it is also fine to be belonged to a single note. A user can make a schedule while posting a new note, or just make a schedule there, so that he can use it for other purposes.

6. Make comments

When users receive data from server of notes that they can see, the status of whether the note can be commented is also sent, so this information will be stored in the front end, if the note is not allowed to be commented, front end will not show the button and text table for comment, vise versa. When a users makes a comment, after pressing the button, texts will be sent to the server, along with other information automatically generated by the front end or the server will be written into the database.

7. Adding filters

Users can set up filters so that they can chose what kind of information they want to see spatially and temporally, the filter can also contain tags and states. But not all the fields are required, user can leave some fields empty.

A user can set up several filters, but only one should be activated at a time. So we created a view activedfilter to store the information of filters that currently activated.

8. Note availabilities

When a user refreshes his interface, like click the refresh button, server will check the filters that currently activated for this user, and select notes accordingly. Then send back the data.

Queries

(1) Create a new user account, with name, login, and password.

```
1  insert into `user`(uid,uname,uemail,upassword) VALUES (4,'u4','u4@nyu.edu','444')
```

信息 概况 状态

[SQL]insert into `user`(uid,uname,uemail,upassword) VALUES (4,'u4','u4@nyu.edu','444')

受影响的行: 1
时间: 0.066s

Result after insertion:

uid	uname	uemail	upassword	ustate	ulongi	ulati
1	u1	u1@nyu.edu	111	at work	32.534167	66.078056
2	u2	u2@nyu.edu	222	lunch	36.948889	66.328611
3	u3	u3@nyu.edu	333	drunk	36.083056	69.0525
4	u4	u4@nyu.edu	444	(Null)	(Null)	(Null)

(2) Add a new note to the system, together with tags, and spatial and temporal constraints.

```
1  INSERT INTO `schedule`(sid,starttime,endtime,repetition)
2  VALUES (4,'2018-11-21 00:00:00','2018-12-31 11:59:59',1);
3
4  INSERT INTO note(nid,ntext,nlongi,nlati,nradius,createdBy,nsid,ntime,visibility)
5  VALUES(4,'text4','36.948889','66.328611',200,3,4,'2018-11-21 00:00:00','all');
6
7  INSERT INTO notetag(tname,nid)
8  VALUES ('tname5',1)
9
```

信息 概况 状态

```
[SQL]INSERT INTO `schedule`(sid,starttime,endtime,repetition)
VALUES (4,'2018-11-21 00:00:00','2018-12-31 11:59:59',1);
受影响的行: 1
时间: 0.030s

[SQL]

INSERT INTO note(nid,ntext,nlongi,nlati,nradius,createdBy,nsid,ntime,visibility)
VALUES(4,'text4','36.948889','66.328611',200,3,4,'2018-11-21 00:00:00','all');
受影响的行: 1
时间: 0.076s

[SQL]

INSERT INTO notetag(tname,nid)
VALUES ('tname5',1)
```

Result after insertion

sid	starttime	endtime	repetition
1	2018-10-03 11:19:42	2018-11-29 11:19:49	0
2	2018-11-26 20:12:26	2018-12-05 20:12:26	1
3	2017-01-01 09:42:46	2018-11-29 09:42:51	7
4	2018-11-21 00:00:00	2018-12-31 11:59:59	1

nid	ntext	ntime	nlongi	nlati	nradius	nsid	createdBy	visibility	commentable
1	note1	2018-11-29 09:47:25	32.534167	66.078056	100	1	1	all	yes
2	note2	2018-11-29 09:47:30	36.948889	66.328611	200	2	2	friends	yes
3	note3	2018-11-29 09:42:03	36.083056	69.0525	300	3	3	self	no
4	text4	2018-11-29 13:33:46	36.948889	66.328611	200	4	3	all	no

tname	nid
tname1	1
tname3	1
tname5	1
tname2	2
tname4	3

(3) For a given user, list all her friends.

```
SELECT uid2 FROM friend WHERE uid1 = 1
```

信息 | 结果1 | 概况 | 状态

uid2
2
3

(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.

```
SELECT * from actived notes an
JOIN note n on an.nid = n.nid
NATURAL JOIN `user`
WHERE an.uid = 1 and `user`.ustate = 'at work'
```

信息 | 结果1 | 概况 | 状态

uid	nid	nBelong	nid1	ntext	ntime	nlongi	nlati	nradius	nsid	createdBy	visibility
1	1	1	1	note1	2018-11-29 09:47:25	32.534167	66.078056	100	1	1	all
1	4	3	4	text4	2018-11-29 13:33:46	36.948889	66.328611	200	4	3	all

(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.

```
SELECT an.uid from actived notes an
join note n on an.nid = n.nid
WHERE an.nid = 4
```

信息 | 结果1 | 概况 | 状态

uid
1
2
3

(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.

```
SELECT * from note n
JOIN actived notes an on an.nid = n.nid
WHERE an.uid = 1 and n.ntext LIKE '%text%'
```

信息 | 结果1 | 概况 | 状态

nid	ntext	ntime	nlongi	nlati	nradius	nsid	createdBy	visibility	commentable	uid	nid1	nBelong
4	text4	2018-11-29 13:33:46	36.948889	66.328611	200	4	3	all	no	1	4	3

Revise our design

Actually, we don't do too much modification about database design. We just tune the engine, which are `MyISAM` before, and the length and type of some attributes.

Implementation

Introduction

We follow the development principles of separation between front-end and back-end.

The front end is mainly based on bootstrap.

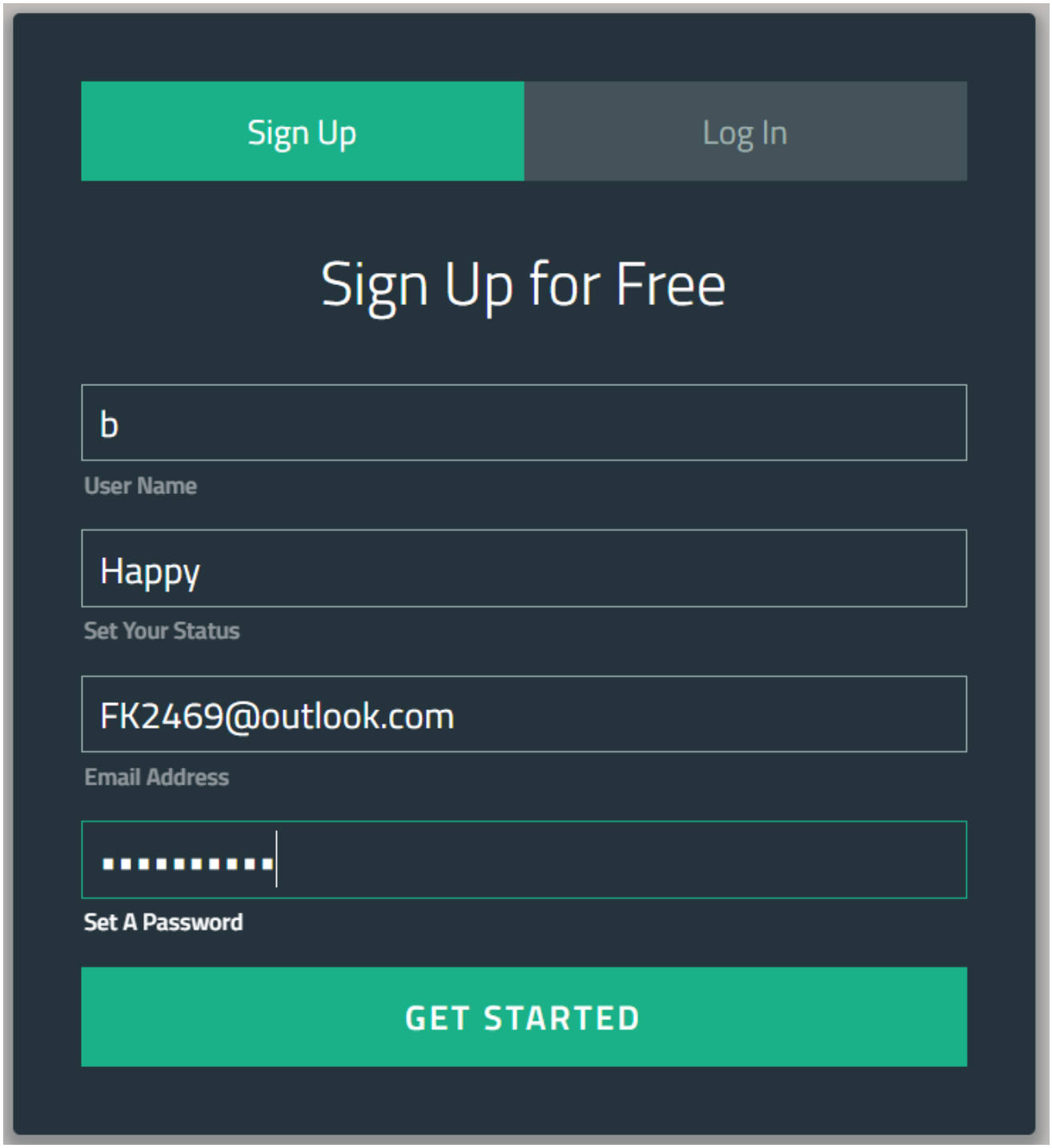
The back end is based on the flask framework, and we use `SQLAlchemy` as our SQL toolkit and Object Relational Mapper (ORM) . With `SQLAlchemy` and some flask extension, our application guard against SQL injection and cross-site scripting attacks.

Run Oingo on your server

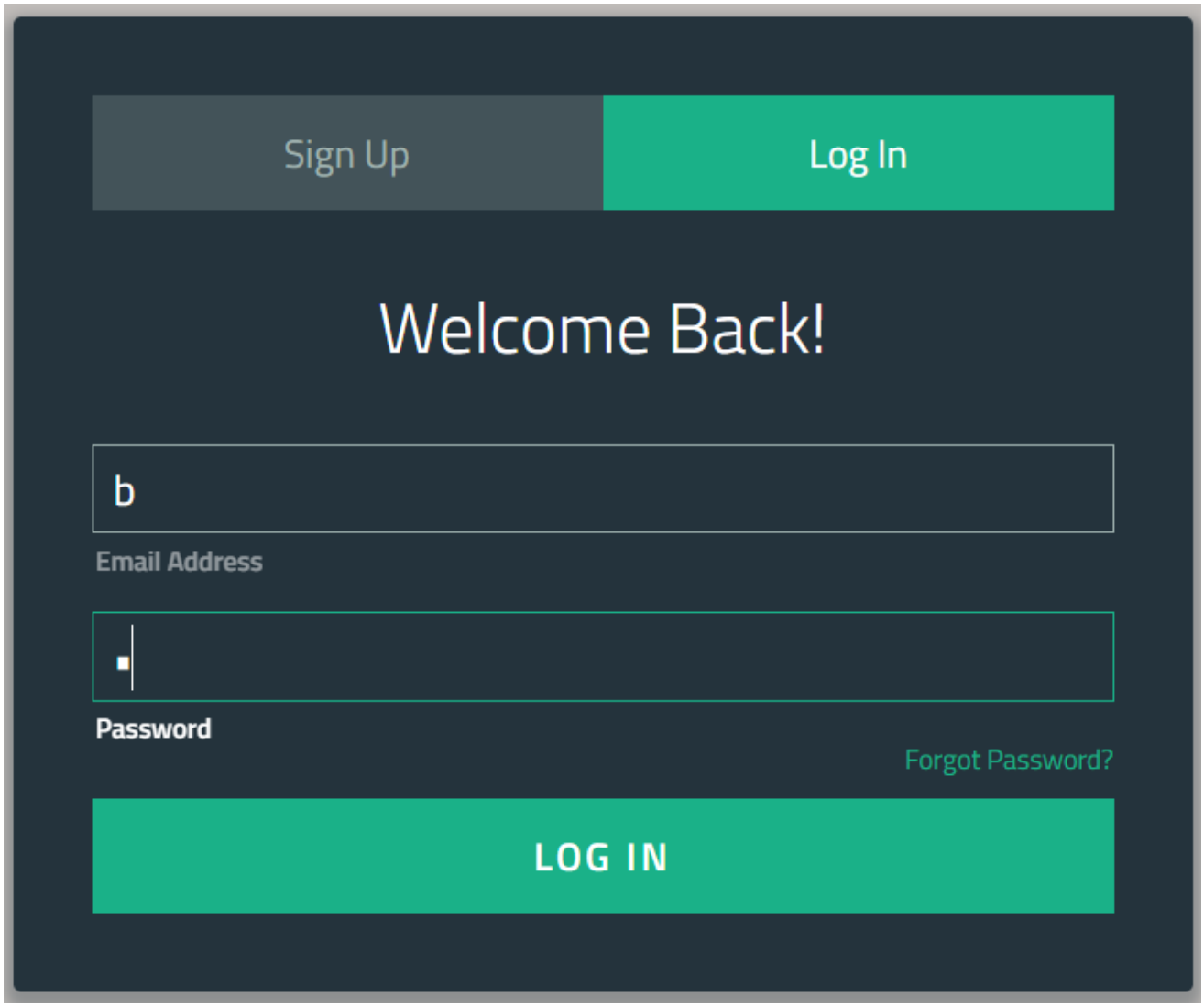
```
      > .\venv\Scripts\activate.ps1
(venv)> pip install -r requirements.txt
(venv)> python app.py
```

Interactive process

Sign Up


Sign Up
Log In
Sign Up for Free
b
User Name
Happy
Set Your Status
FK2469@outlook.com
Email Address
Set A Password
GET STARTED

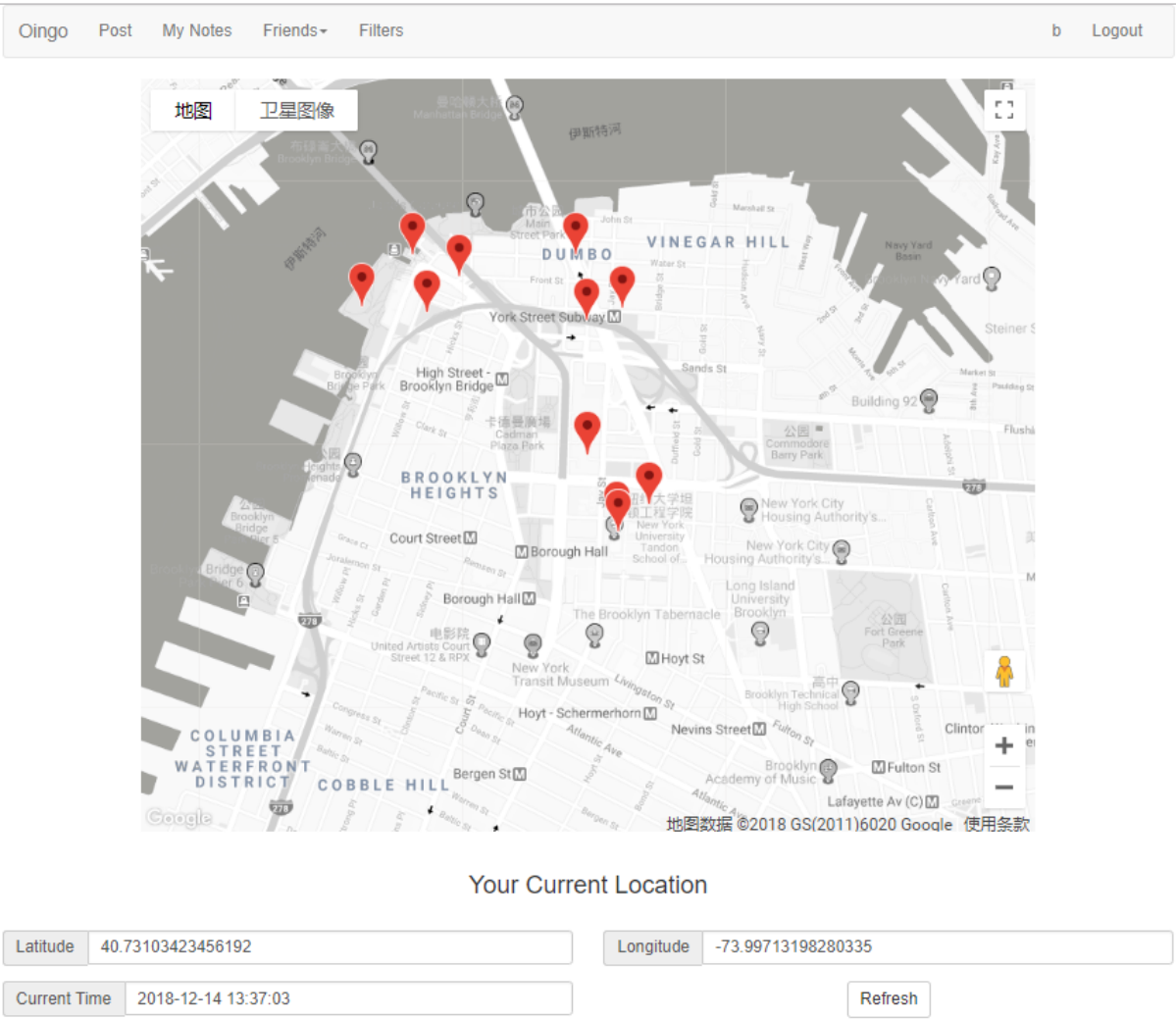
Login



Timeline

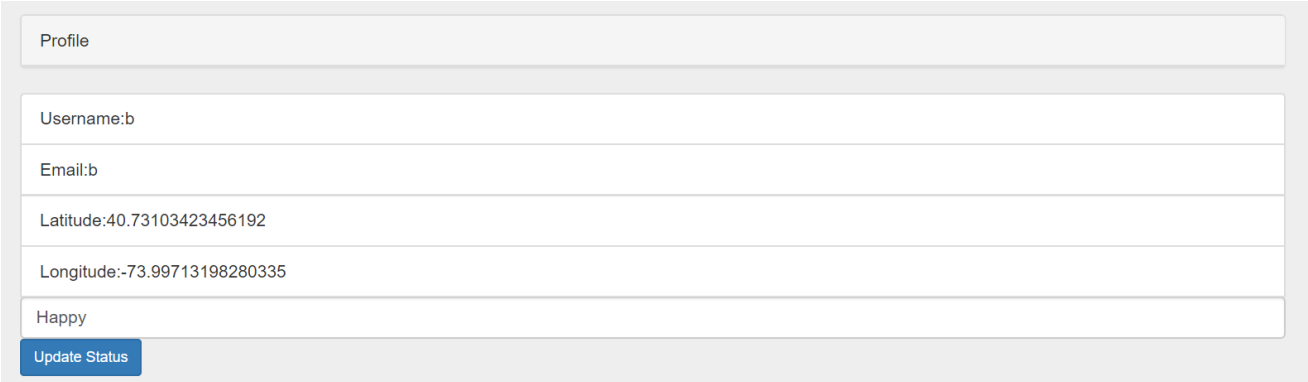
On this page, you can see all notes, which are filtered by your filter and are activated now.

TA can use three input box below to change the Time and Location of current user.

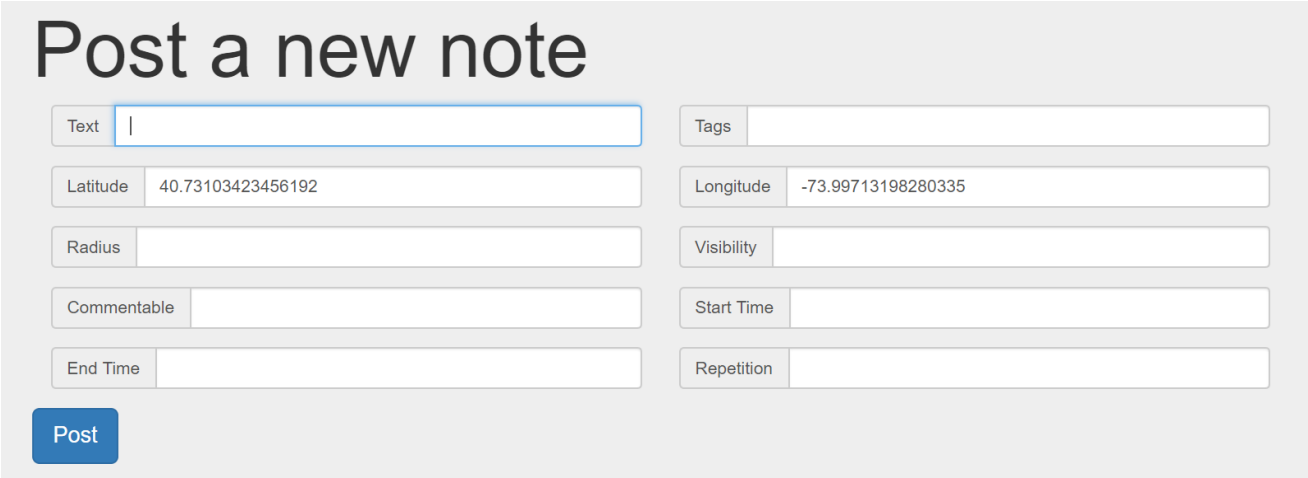


Profile

If you click username, you can see the profile of specified user, say X. If you are X, then you can update your status through this page. Or you can't see the button `Update Status`.

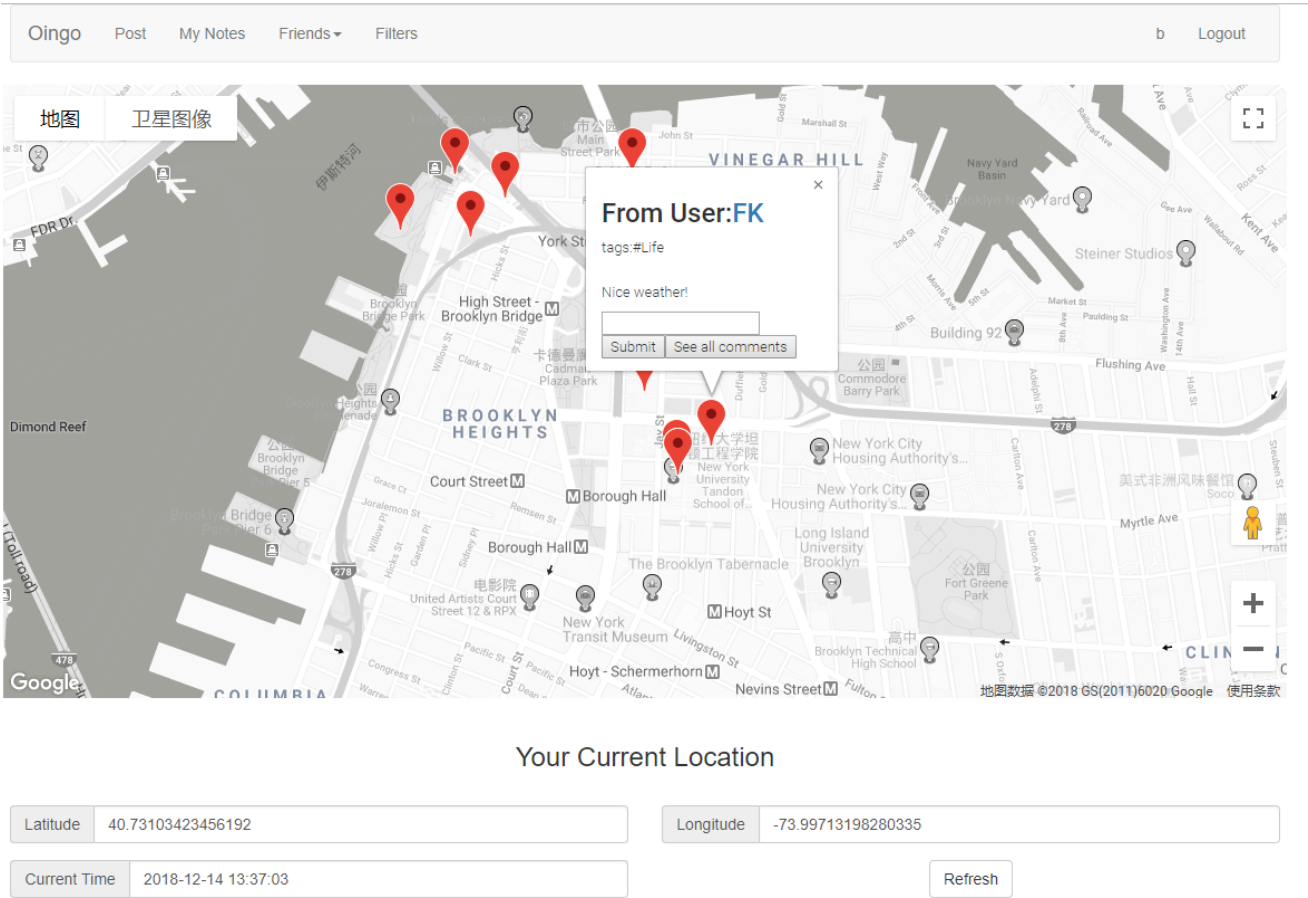


Post



Click one note on Map

If you click a specified note on Map, you can see its detail. And you can comment if you are allowed .



My Notes

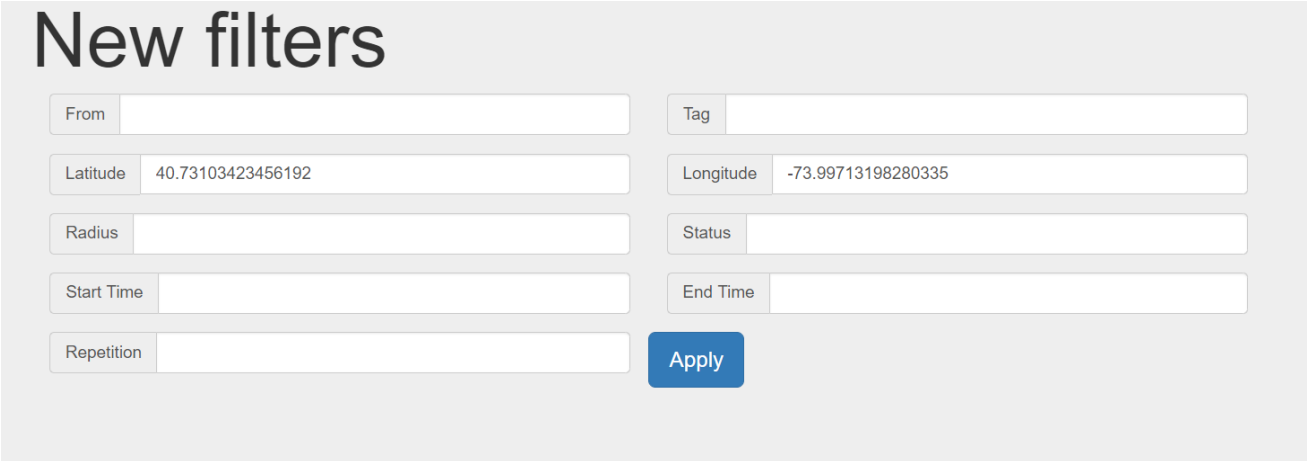
You can see all notes you created on this page. And you can delete them, of course.

My Notes

Note ID	Text	Tag	Commentable	Visibility	Action
5	What a cute squirrel!	#Journey,#Life	0	self	Delete
24	I am in the Empire State Building	#Journey,#Life	1	all	Delete
25	Washington Square is a good place!	#Journey,#Life	1	all	Delete

Filters

You can add new filter and see all your filters on this page. By the way, you can click the button Disable Or Active to decide whether you will use this filter now.

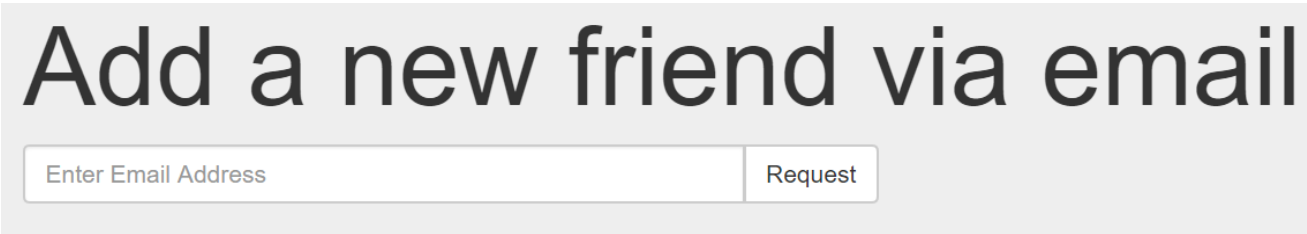


My filters

Filter ID	From	Latitude	Longitude	Radius	State	Tag	Time	Filter uid	Action
1	friend	40.69593543552976	-73.9845148715973	30000	Happy	#Life	2018-12-14 12:13:04	3	Disable
2	self	40.69593543552976	-73.9845148715973	30000	Happy	#Life	2018-12-14 12:13:04	3	Disable

Add new friend

You can add a new friend through this page. You should type the Email Address of the user who you want to make friend with.



Friend requests

On this page, you can see all requests you received, and decide whether accept them.

Friend Requests

Email	Id	Latitude	Longitude	Name	State	Accept
FK2469@outlook.com	1	40.696173	-73.982861	FK	CSing	Accept
FK@outlook.com	7	40.696173	-73.982860	c	A	Accept

Friend list

On this page, you can see all your friends, and you can delete them, which means, you are not friend anymore.

Friend List

Email	Id	Latitude	Longitude	Name	State	Action
FK2469@outlook.com	1	40.696173	-73.982861	FK	CSing	Delete
A	2	40.696173	-73.982861	A	A	Delete
FK@outlook.com	7	40.696173	-73.982860	c	A	Delete
F469@outlook.com	8	123	123	FK	CSing	Delete
F49@outlook.com	9	123	123	FK	CSing	Delete
F@outlook.com	10	123	123	FK	CSing	Delete
qeqw@qq.com	14	40.697627368267106	-73.98758331871034	asdf	sadfa	Delete

API Design

Register

```
app.route('/register', methods=['POST'])

{
    "uname": "FK",
    "uemail": "FK2469@outlook.com",
    "upassword": "123456",
    "ustate": "CSing",
    "ulongi": "123",
    "ulati": "123"
}
```

Login

```
@app.route('/login', methods=['POST'])

{
    "uemail": "b",
    "upassword": "b"
}
```

Logout

```
@app.route('/logout', methods=['POST'])

{
    "uemail": "b"
}
```

UpdateUserCurrentLocation

```
@app.route('/updateUserLocation', methods=['POST'])

{
    "uemail": "FK2469@outlook.com",
    "ulongi": "456",
    "ulati": "456"
}
```

UpdateUserCurrentTime

```
@app.route('/updateUserCurrentTime', methods=['POST'])

{
    "uemail": "b",
    "ucurrentTime": "2018-12-25 11:08:19"
}
```

UpdateUserCurrentState

```
@app.route('/updateUserState', methods=['POST'])

{
    "uemail": "b",
    "ustate": "Working on Project"
}
```

Get current user's timeline

```
@app.route('/notes/timeline', methods=['GET'])

{
    "uemail": "b"
}

RESPONSE EXAMPLE:

{
    "notesList": [
        {
            "commentable": "1",
```

```
            "createdBy": 3,
            "nid": 1,
            "nlati": "40.695872",
            "nlongi": "-73.983345",
            "nradius": 60,
            "nsid": 3,
            "ntext": "A",
            "ntime": "Tue, 11 Dec 2018 10:21:50 GMT",
            "tag": [
                "#DB",
                "#FK",
                "#JX"
            ],
            "visibility": "all"
        },
        {
            "commentable": "0",
            "createdBy": 3,
            "nid": 2,
            "nlati": "40.695872",
            "nlongi": "-73.983345",
            "nradius": 100,
            "nsid": 1,
            "ntext": "B",
            "ntime": "Tue, 11 Dec 2018 09:11:15 GMT",
            "tag": [
                "#DB"
            ],
            "visibility": "friend"
        }
    ]
}
```


Get all notes created by current user


```
@app.route('/notes', methods=['GET'])
{
    "uemail": "b"
}

{
    "notesList": [
        {
            "commentable": "1",
            "createdBy": 3,
            "nid": 1,
            "nlati": "40.695872",
            "nlongi": "-73.983345",
            "nradius": 60,
            "nsid": 3,
            "ntext": "A",
            "ntime": "Tue, 11 Dec 2018 10:21:50 GMT",
            "tag": [
```

```
                "#DB",
                "#FK",
                "#JX"
            ],
            "visibility": "all"
        },
        {
            "commentable": "0",
            "createdBy": 3,
            "nid": 2,
            "nlati": "40.695872",
            "nlongi": "-73.983345",
            "nradius": 100,
            "nsid": 1,
            "ntext": "B",
            "ntime": "Tue, 11 Dec 2018 09:11:15 GMT",
            "tag": [
                "#DB"
            ],
            "visibility": "friend"
        }
    ]
}
```

Add a new note

```
@app.route('/notes', methods=['POST'])

{
    "uemail": "b",
    "ntext": "Thank you TS!",
    "tag": [
        "#DB"
    ],
    "ntime": "2018-12-11 11:08:19",
    "nlongi": "123",
    "nlati": "31",
    "nradius": "100",
    "visibility": "all",
    "commentable": "1",
    "starttime": "2018-12-01 10:08:19",
    "endtime": "2018-12-30 10:08:19",
    "repetition": "000000000"
}
```

Delete specified note

```
@app.route('/notes/<int:nid>', methods=['DELETE'])

{
    "uemail": "b"
}
```

Update specified note

```
@app.route('/notes/<int:nid>', methods=['PUT'])

{
    "uemail": "b",
    "ntext": "Fuck TS!",
    "tag": [
        "#DB"
    ],
    "ntime": "2018-12-11 11:08:19",
    "nlongi": "123",
    "nlati": "31",
    "nradius": "100",
    "visibility": "all",
    "commentable": "1",
    "starttime": "2018-12-01 10:08:19",
    "endtime": "2018-12-30 10:08:19",
    "repetition": "000000000"
}
```

Add a new filter

```
@app.route('/filters', methods=['POST'])

{
    "uemail": "FK2469@outlook.com",
    "ffrom": "self",
    "ftag": "#TS",
    "flongi": "121",
    "flati": "121",
    "fradius": "10",
    "ftime": "2018-12-11 19:26:04",
    "fstate": "CSing",
    "starttime": "2018-12-01 10:08:19",
    "endtime": "2018-12-25 10:08:19",
    "repetition": "0000000"
}
```

Delete specified filter

```
@app.route('/filters/<int:fid>', methods=['DELETE'])

{
    "uemail": "b"
}
```

Update specified filter

```
@app.route('/filters/<int:fid>', methods=['PUT'])
```

```
{
    "uemail": "c",
    "ffrom": "self",
    "ftag": "#TS",
    "flongi": "121",
    "flati": "121",
    "fradius": "10",
    "ftime": "2018-12-11 19:26:04",
    "fstate": "Boring",
    "starttime": "2018-12-01 10:08:19",
    "endtime": "2018-12-25 10:08:19",
    "repetition": "00000010"
}
```

Get all filters of current user

```
@app.route('/filters', methods=['GET'])

{
    "filtersList": [
        {
            "ffrom": "self",
            "fid": 1,
            "flati": "121",
            "flongi": "121",
            "fradius": 10,
            "fsid": 4,
            "fstate": "Boring",
            "ftag": "#TS",
            "ftime": "Tue, 11 Dec 2018 19:26:04 GMT",
            "fuid": 3
        },
        {
            "ffrom": "all",
            "fid": 2,
            "flati": "40.696173",
            "flongi": "-73.982861",
            "fradius": 10,
            "fsid": 1,
            "fstate": "CSing",
            "ftag": "#DB",
            "ftime": "Tue, 11 Dec 2018 12:56:58 GMT",
            "fuid": 3
        }
    ]
}
```

Get all comments of specified note

```
@app.route('/notes/<int:nid>/comments', methods=['GET'])

{
```

```
    "commentsList": [
        {
            "cid": 1,
            "createdBy": 2,
            "ctext": "A",
            "nid": 1
        },
        {
            "cid": 2,
            "createdBy": 3,
            "ctext": "B",
            "nid": 1
        },
        {
            "cid": 3,
            "createdBy": 2,
            "ctext": "C",
            "nid": 1
        },
        {
            "cid": 4,
            "createdBy": 1,
            "ctext": "Cool!!!!",
            "nid": 1
        }
    ]
}
```

Add comment on specified note

```
@app.route('/notes/<int:nid>/comments', methods=['POST'])

{
    "uemail": "b",
    "ctext": "TATS"
}
```

Delete specified comment (Note creator and Comment creator only)

```
@app.route('/comments/<int:cid>', methods=['DELETE'])

{
    "uemail": "b"
}
```

Update specified comment (Comment creator only)

```
@app.route('/comments/<int:cid>', methods=['PUT'])

{
    "uemail": "b",
    "ctext": "adnf"
}
```

Send friend request to someone

```
@app.route('/request', methods=['POST'])

{
    "uemail": "b",
    "sendTo": "FK@outlook.com"
}
```

Get all received friend requests

```
@app.route('/requests', methods=['GET'])

{
    "uemail": "b"
}

RESPONSE EXAMPLE

{
    "requestsList": [
        {
            "uemail": "A",
            "uid": 2,
            "ulati": "40.696173",
            "ulongi": "-73.982861",
            "uname": "A",
            "ustate": "A"
        },
        {
            "uemail": "FK2469@outlook.com",
            "uid": 1,
            "ulati": "40.696173",
            "ulongi": "-73.982861",
            "uname": "FK",
            "ustate": "CSing"
        }
    ]
}
```

Accept friend request

```
@app.route('/requests/<int:requestFrom>', methods=['POST'])

{
    "uemail": "b"
}
```

GET my friend lists

```
@app.route('/friends', methods=['GET'])

{
    "uemail": "b"
}

{
    "friendsList": [
        {
            "uemail": "FK2469@outlook.com",
            "uid": 1,
            "ulati": "40.696173",
            "ulongi": "-73.982861",
            "uname": "FK",
            "ustate": "CSing"
        },
        {
            "uemail": "A",
            "uid": 2,
            "ulati": "40.696173",
            "ulongi": "-73.982861",
            "uname": "A",
            "ustate": "A"
        }
    ]
}
```