

# Project Report

Oingo

Project Title: \_\_\_\_\_

Project Objective:

Initial Project

The final version of Oingo is a working system and it meets all functional requirements.

Agreed Changes to Project Objective:

The final version of Oingo is a working system that includes a user-friendly user interface at frontend and reliable backend. The system is also secure and immune to common attacks.

Project Overview:

The system is accessed by users with any internet web browser. The users can publish information in the form of short notes, linked with locations and certain times, attached some tags. And other users can receive these notes on their own location, time, and some filters they may set. The system utilizes Django framework. All the information associated with users is stored in our server.

Status of the Project and Handover to:

Status:	Handover To:
Fully working product, all requirements are met. Following iterations will mainly focus on maintenance and possible bugs reported by users.	Prof. Suel

Prepared by:

Guanwei Zhang, Chris Fan

## Introduction

Oingo is a new mobile app that allows users to share useful information via their mobile devices based on social, geographic, temporal, and keyword constraints. The main idea in Oingo is that users can publish information in the form of short notes, linked with locations and certain times, attached some tags. And other users can receive these notes on their own location, time, and some filters they may set.

## Design & Explanation

### 1. Proposed Functionalities

#### a. User Registration & Authentication

The system should have a page for new users to sign up their account by supplying an email address, a username, and a password. After sign-up, a profile should be created automatically, and user should be able to modify the profile later.

The system should also have a page for user to login with their registered email and password. The user should access to personalized contents such as their own notes, profile, friend list, filters, etc. after and only after login.

#### b. Profile Management

The system should provide the login user with access to manage their own profile. The profile should at least consist of username, password, email, state, and last recorded location.

The system should allow user to modify their username, password and state, while not allow them to modify email which serves as an identifier in login process.

For newly created profile, user's state should be assigned with a default value. During modification, there should be some default states for user to choose, and user should also be able to create their own state.

The last recorded location shouldn't be modified manually. The system should collect it automatically every time the user performs actions such as logging in, posting notes, etc. The system should remove previous records and only maintain the last recorded location considering privacy issues.

c. Friendship Management

The system should maintain a friend list for each user. A user should be able to send friend request to another user. Once the other user accepts the request, the two users would have each other on their own friend list. In the current stage, it shouldn't be a complex networking system supporting all kinds of networking operations. It should be a simple one that just letting users to follow each other.

d. Note Posting

The system should provide interfaces for user to post their notes. A standard note should have a description, a schedule indicating time and frequency the note could be seen, a location indicating the place, a radius within which the note could be seen, and some tags specifying what kind of note it is.

When publishing a note, user should also decide by what group of people the note could be seen, e.g. only by themselves, only by friends, or maybe by everyone; And whether the note allows others to comment.

Besides, the system should create a timestamp to indicate the publish time.

e. Comment Management

The system should allow user to comment on notes they receive if the note is set so. A comment should consist of content and a timestamp.

In this stage, we don't make it too complex such as with a whole bunch of communication system, it should be as simple as attaching some words to the note.

f. Filtering Management

The system should have a page for user to create and customize filters to decide when and where to receive what kind of notes, under which state. A filter should have a conjunction of conditions, such as on schedule, on location, on tags, and on user state.

User could create several filters. Different filters should be in active at same time, and the final filtered result shown to user should be the union of the results from each filter.

g. Note Delivery

The system should deliver filtered notes to certain users. A user should receive a note if and only if:

- User is in the visible group of the note, and current location and time are matched to the note's location radius and schedule, and

- And, there should be at least one filter matching the user's state, at least one tag in the notes, and matching the current location and time.

#### h. Area Management

The system should maintain larger area information other than just locations. For example, a location could be a restaurant, a park, a shop, etc. And an area should be larger districts such as SoHo, Brooklyn, Flushing, etc. A location belongs to zero or more area, and an area contains zero or more locations.

### 2. Entity description

#### a. User

User entity should be responsible to user profiles. It should contain:

- Username.
- Password.
- Email also served as login.
- User state, user can select from default states, and create new ones.
- User's last recorded location recorded by the system automatically.

#### b. Note

Note entity should deal with notes information, including:

- Author: the user who posted the note
- Note content: whatever the user wants to say in the note, should be limit to a short one.
- Publish time: the timestamp indicating when the note is posted.
- Visible group: to whom it is visible, using numbers to indicate. different groups, such as 0 for user themselves, 1 for friends, and 2 for everyone.
- Allow comment: true if user can comment on the note, or false if not.
- Visible radius: users could only see the note within a range indicating by the radius.

#### c. Schedule

Schedule entity should maintain scheduling information about notes:

- From and end date: the note could be only valid during this period.
- Repetition: days of the cycle the note becomes visible, e.g. 1 for one day, 7 for a week, and 365 for a year, etc.
- Start and end time: a time range specifying during which time in each cycle the note could become visible.

d. Location

Location entity should have information of a place. A location could be already in the database, and it could also be created by the user when creating a note. It should have:

- Location name: name of the location.
- Longitude and latitude: indicating the coordinate of the location.

e. Tag

Tag entity should collect all the tags the user created, and maybe some default tags as well.

In current stage, it has only one attribute tag name indicates what tag it is.

f. Filter

Filter entity should have information about what kind of note should be received by the user. So, it should contain:

- Start and end time
- From and end date
- Repetition
- Location name
- Longitude and latitude
- Radius
- User state
- Tag

A filter is a conjunction of those attributes above. These attributes specifying conditions could be null, which indicates the condition don't matter in this filter. Based on users' choices, there should be zero or more filters active at the same time.

g. Area

To decide whether a location belongs to it, Area entity should specify:

- the area's center longitude and latitude
- and its radius

In our plan, besides calculating by coordinates, an area should also contain some special locations not in the radius, with the help of Belong relation.

For example, we may define an area representing America. The radius might only cover the continent locations. But we can make a location stands for Hawaii belongs to it.

3. Relation description

a. Friendship

There should be a relation Friendship between different users. A user could send a friending request to another user, and if the other user accept the request, then a Friendship created to describe this relation.

Ideally Friendship is bidirectional, i.e. user A is a friend of user B, so user B must be a friend of user A. But in some situation, A may want to remove B from friendship silently while B still follows A, or maybe A is an online celebrity and many fans want to follow A. In such situations, friendship could be unidirectional (like follow in twitter). In current stage, we don't concentrate too much on such issues.

There should be a table for Friendship to describe a many-to-many relation. This can handle the both situations talked above.

b. Publish

User and Note should have a relation Publish, indicating user published a note. This should be a one-to-many relation because user may publish many notes, but one note could be only published by one user. Also, Note is totally participating the relation since a note only exists after a user publishes it.

To achieve this relation, there should be a foreign key in Note referencing user id, indicating which user has published the note.

c. Comment

User and Note should also have another relation Comment, which specifies user make a comment on a note. A comment should have content and a timestamp, and specify which user commented on which note. To achieve this relation, there should be a table for Comment to describe the many-to-many this relationship with extra attributes.

d. With

Note and Schedule should have a relation called With, indicating a note is set with a schedule.

At design phase, we plan this as a one-to-one relation, since one note should have only one schedule, and a schedule only relates to one note.

To achieve this, there should be a foreign key referencing schedule id, indicating which schedule the note is set with.

In the future work, user may want some frequently-used schedules saved so that they needn't set a schedule repeatedly. This makes the relation to be one to many, but won't affect our schema with foreign key solution, and we can safely extend such functions without modification to the schema.

e. In

Note and Location should have a relation In, indicating a note is set in a location.

This should be a one-to-many relation since a note only has one location while there may be many notes posted in one same location.

To achieve this relation, there should be a foreign key in Note.

f. Attach

Note and Tag should have a relation called Attach, indicating a note is attached with a tag.

A note should be attached with zero or more tags and a tag should be attached to zero or more notes. So, this should be a many-to-many relation.

To achieve this, there should be a table for Attach.

g. Set

User and Filter should have a relationship called set; users can set filters of their own interests. The relation indicates a user is associated with his/her filter settings.

A user can set up 0 to multiple filters, every filter must have a user associated with it, hence it is a total partition relation between User and Filter.

h. Include

Filter and Tag should have a relationship called Include, a filter can have a tag included and the tag is set up by the corresponding user.

A filter is explicitly designed to have 1 tag associated with it for convenience. Filters with multiple tags can be achieved by using conjunction of multiple filters with different tags. Since different filters can use same tag, the relationship between Filter and Include is many-to-one relation.

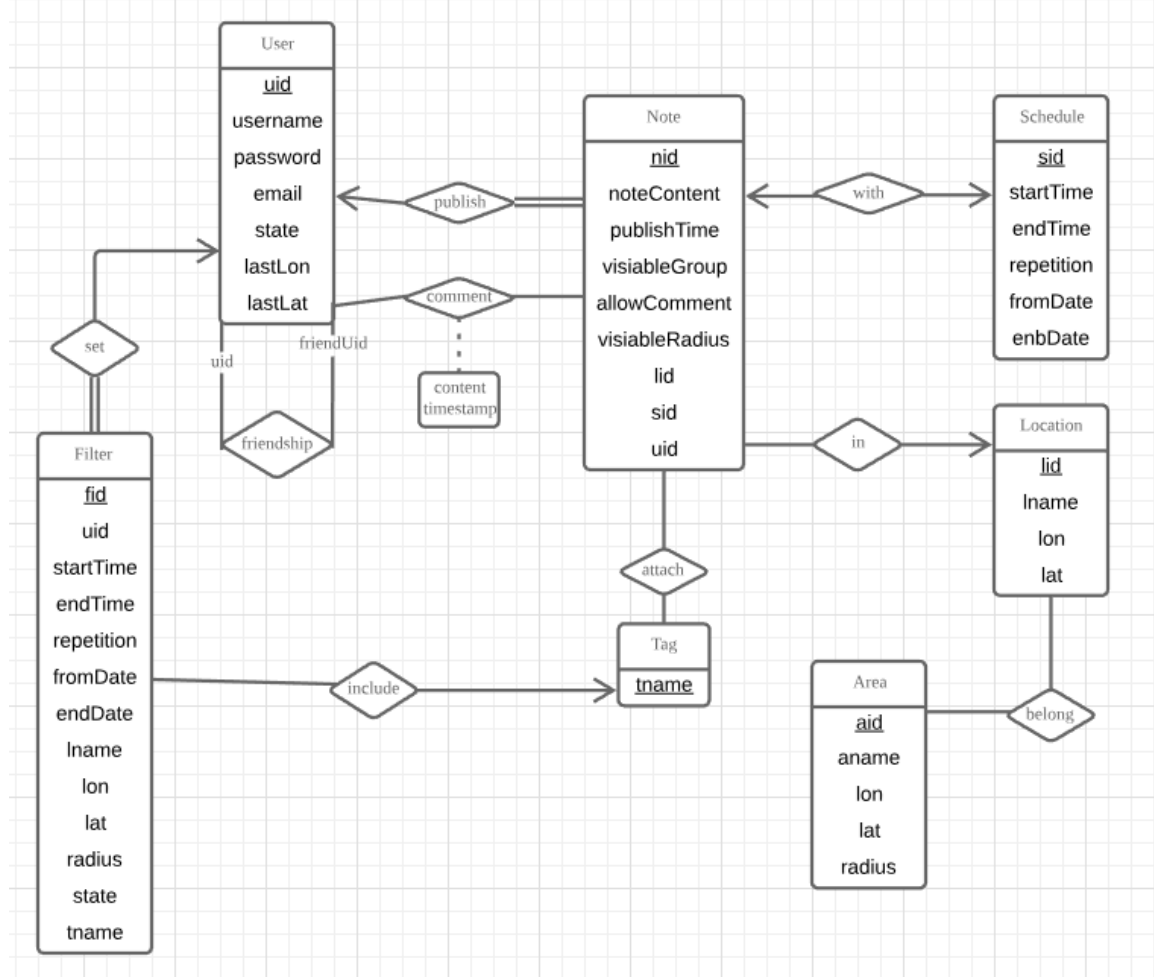
i. Belong

Location and Area should have a relationship called Belong; locations belong to an area and that is common sense (e.g. multiple restaurants in SOHO belong to area SOHO). The relationship indicates locations are associated with areas.

In an area, there can be multiple locations since locations are implemented as points identified by longitude and latitude and areas are identified by the center (also a point) and approximate radius. It is a many-to-many relation because a location can also belong to multiple areas (e.g. a restaurant in area SOHO also belong to area Lower Manhattan).

## ER Diagram

According to the design we described above, we may create a DR diagram below:



## Schema & Constraints

Based on the ER diagram, we create the following schema:

- User(uid, username, password, email, state, last\_lon, last\_lat)
- Note(nid, uid, lid, sid, noteContent, publishTime, visibleRadius, visibleGroup, allowComment)
- Schedule(sid, startTime, endTime, repetition, fromDate, endDate)
- Location(lid, lname, lon, lat)
- Tag(tname)
- Filter(fid, uid, tname, startTime, endTime, repetition, fromDate, endDate, lname, lon, lat, radius, state)
- Area(aid, aname, lon, lat, radius)
- Friendship(uid, friendUid)
- Comment(uid, nid, content, timestamp)



- Attach(nid, tname)
- Belong(lid, aid)

Foreign Keys:

Note.uid is a foreign key referencing User.uid

Note.lid is a foreign key referencing Location.lid

Note.sid is a foreign key referencing Schedule.sid

Comment.uid is a foreign key referencing User.uid

Comment.nid is a foreign key referencing Note.nid

Schedule.nid is a foreign key referencing Note.nid

Attach.nid is a foreign key referencing Note.nid

Attach.tname is a foreign key referencing Tag.tname

Belong.lid is a foreign key referencing Location.lid

Belong.aid is a foreign key referencing Area.aid

Friendship.uid is a foreign key referencing User.uid

Friendship.friendUid is a foreign key referencing User.uid

Filter.uid is a foreign key referencing User.uid

Filter.tname is a foreign key referencing Tag.tname

## Tests

### 1. Test data

User:

uid	username	password	email	state	last_lon	last_lat
100001	Antonio Rod	Antonio123	AntonioRod@gmail.com	at work	-73.9776	40.6837
100002	Jerry Tong	Jerry123	JerryTong@gmail.com	just chilling	-74.0114	40.7065
100003	Brown Snow	Brown123	BrownSnow@hotmail.com	just chilling	-73.9997	40.7093
100004	Guru Singh	Guru123	GuruSingh@gmail.com	lunch break	-122.413	37.7875
100005	Ivy Yu	Ivy123	IvyYu@hotmail.com	at work	-71.0698	42.3533
100006	Jack Smith	Jack123	JackSmith@gmail.com	lunch break	-73.9786	40.7573
100007	Jane Jing	Jane123	JaneJing@gmail.com	just chilling	-79.6568	43.5878
100008	Tom White	Tom123	TomWhite@gmail.com	on vacation	-122.325	47.6066
100009	Jose Luis	Jose123	JoseLuis@gmail.com	just chilling	-99.1318	19.4165
100010	Feihong Huang	Feihong123	FeihongHuang@qq.com	just chilling	121.101	31.144

Note:

nid	uid	lid	sid	noteContent	publishTime	visibleRadius	visibleGroup	allowComment
10001	100003	102	1004	Food is delicious!!!	2018-10-05 13:00:00	100	2	1
10002	100002	101	1003	MoMa is awesome	2018-06-08 15:05:32	100	1	1
10003	100004	104	1001	WTC	2018-05-14 16:43:54	100	0	1
10004	100001	105	1002	Had a good dinner	2017-08-02 19:54:23	100	2	1
10005	100009	103	1003	Delicious food and resonable price	2018-03-21 14:12:06	100	2	1
10006	100010	108	1006	Awesome view up here	2018-10-11 20:06:43	100	2	1
10007	100008	106	1007	Walgreens is packed now	2018-07-19 13:34:34	100	2	1
10008	100006	107	1008	Busy palce	2018-05-17 12:45:50	100	2	1

### Schedule & Location:

sid	startTime	endTime	repetition	fromDate	endDate	lid	lname	lon	lat
1001	12:00:00	20:00:00	1	2017-10-05	2019-10-05	101	MoMA	-73.977	40.761
1002	11:00:00	12:00:00	1	2017-10-05	2019-11-05	102	Cafe China	-73.9821	40.7501
1003	09:00:00	12:00:00	2	2017-10-05	2019-11-05	103	Taiwan Pork Chop House	-73.998	40.7142
1004	15:00:00	15:30:00	3	2017-10-05	2019-11-05	104	World Trade Center	-74.0134	40.7127
1005	16:00:00	18:00:00	4	2017-10-05	2019-11-05	105	Wild Ginger Seattle	-122.337	47.6088
1006	19:00:00	19:45:00	5	2017-10-05	2019-11-05	106	Walgreens	-122.412	37.8015
1007	20:00:00	20:30:00	6	2017-10-05	2019-12-05	107	McDonalds	-99.1554	19.425
1008	09:00:00	10:00:00	7	2017-10-05	2019-12-05	108	Dong Fang Ming Zhu Dian Shi Ta	121.495	31.2419
1009	08:00:00	08:30:00	8	2017-10-05	2019-12-05				

### Tag & Area:

tname	aid	aname	lon	lat	radius
Dinner	11	China Town, NY	-73.9965	40.7161	2000
Lunch	12	Midtown Manhattan	-73.9841	40.7549	5000
McDonalds	13	Lower Manhattan	-73.9998	40.7218	5000
Michelin	14	Seattle	-122.332	47.6063	80000
NYC	15	Shanghai	121.474	31.2303	100000
Sightseeing	16	Fisherman Wharf, San Francisco	-122.418	37.808	1500
Walgreens	17	Mexico City	-99.1333	19.4325	100

### Filter:

fid	uid	tname	startTime	endTime	repetition	fromDate	endDate	lname	lon	lat	radius	state
1	100001	NYC	10:00:00	14:00:00	10	2018-10-05	2018-11-05	MoMA	-73.977	40.761	1000	just chilling
2	100002	NYC	10:00:00	15:00:00	20	2018-11-05	2018-12-05	World Trade Center	-74.0134	40.7127	1000	just chilling
3	100003	Lunch	09:00:00	13:00:00	30	2018-10-06	2018-12-05	Taiwan Pork Chop House	-73.998	40.7142	1000	just chilling
4	100004	Dinner	08:00:00	18:00:00	100	2018-09-05	2019-10-16	Cafe China	-73.9821	40.7501	1000	just chilling
5	100005	Sightseeing	09:00:00	16:00:00	365	2018-05-14	2020-09-24	Dong Fang Ming Zhu Dian Shi Ta	121.495	31.2419	1000	just chilling
6	100006	Michelin	13:00:00	15:00:00	50	2018-04-13	2018-10-05	McDonalds	-99.1554	19.425	1000	just chilling
7	100007	NYC	16:00:00	19:00:00	10	2018-01-28	2018-06-14	World Trade Center	-74.0134	40.7127	1000	just chilling
8	100008	Lunch	12:00:00	13:00:00	20	2018-11-12	2019-10-05	Walgreens	-122.412	37.8015	1000	just chilling

### Friendship & Attach & Belong:

uid	friendUid	nid	tname	lid	aid
100008	100001	10004	Dinner	103	11
100001	100002	10001	Lunch	102	12
100004	100002	10005	Lunch	101	13
100001	100003	10008	McDonalds	104	13
100002	100003	10002	NYC	105	14
100002	100005	10003	NYC	108	15
100003	100005	10005	NYC	106	16
100005	100006	10007	Sightseeing	107	17
100006	100007	10007	Walgreens		
100007	100008				

### Comment:

uid	nid	content	timeStamp
100001	10001	useful, food is delicious	2018-06-08 15:05:32
100002	10003	WTC is a nice place, will come back next time	2018-06-08 15:05:32
100003	10002	useful, nice collections in MoMA	2018-06-08 15:05:32
100004	10008	useful, it is busy and the service is average	2018-06-08 15:05:32
100005	10007	useful, it is packed	2018-06-08 15:05:32
100007	10005	useful, their pork chop is the best	2018-06-08 15:05:32
100008	10006	useful, the architecture is spectacular	2018-06-08 15:05:32
100010	10004	useful, Wild Ginger is a good place to have lunch	2018-06-08 15:05:32

2. Query Tasks (with placeholder):

- a. Create a new user account, with name, login, and password.

```
insert into User(username, password, email, state, last_lon, last_lat)
values (@username, @password, @email, @default_state, @fake_lon, @fake_lat);
```

- b. Add a new note to the system, together with tags, and spatial and temporal constraints.

```
insert ignore into Tag(tname)
values (@tname1), (@tname2);
insert into Schedule(startTime, endTime, repetition, fromDate, endDate)
values (@startTime, @endTime, @repetition, @fromDate, @endDate);
select LAST_INSERT_ID() into @v_sid;
insert into Location(lname, lon, lat)
values (@lname, @lon, @lat);
select LAST_INSERT_ID() into @v_lid;
insert into Note(noteContent, publishTime, visibleGroup,
allowComment, visibleRadius, lid, sid, uid)
values (@noteContent, @publishTime, @visibleGroup, @allowComment,
@visibleRadius, @v_lid, @v_sid, @uid);
select LAST_INSERT_ID() into @v_nid;
insert into Attach(nid, tname)
values (@v_nid, @tname1), (@v_nid, @tname2);
```

- c. For a given user, list all her friends.

```
select friendUid from Friendship where uid=@uid;
```

- d. 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.

Should define this function for (d) and (e):

```
/*
DELIMITER $$
CREATE FUNCTION `haversine`(
    lat1 FLOAT, lon1 FLOAT,
    lat2 FLOAT, lon2 FLOAT
) RETURNS float
NO SQL
DETERMINISTIC
COMMENT 'Returns the distance in degrees on the Earth between two points.
To get miles, multiply by 3961, and km by 6373'
BEGIN
    RETURN DEGREES(ACOS(
        COS(RADIANS(lat1)) *
        COS(RADIANS(lat2)) *
        COS(RADIANS(lon2) - RADIANS(lon1)) +
        SIN(RADIANS(lat1)) * SIN(RADIANS(lat2))
    ));
END;
DELIMITER;
*/
```

```
with
open_notes as (
select * from Note natural join Schedule natural join Location
where (uid=@uid
      or (visibleGroup=1 and uid in (select uid from Friendship where friendUid=@uid))
      or visibleGroup=2)
and (
  startTime is null or endTime is null
  or Time(@currentTime) between startTime and endTime
)
and (
  fromDate is null or endDate is null or repetition is null
  or (Date(@currentTime) between fromDate and endDate
  and datediff(Date(@currentTime),fromDate) % repetition=0)
)
and (
  visibleRadius is null or lon is null or lat is null
  or (visibleRadius >= haversine(lat, lon, @currentLat, @currentLon)*6373)
)
)

select distinct nid from
open_notes n natural left outer join Attach a, `Filter` f
where f.uid=@uid
and (f.state is null or f.state=@state)
and (f.tname is null or f.tname=a.tname)
and (
  f.startTime is null or f.endTime is null
  or Time(@currentTime) between f.startTime and f.endTime
)
and (
  f.fromDate is null or f.endDate is null or f.repetition is null
  or (Date(@currentTime) between f.fromDate and f.endDate
  and datediff(f.fromDate, Date(@currentTime)) % f.repetition=0)
)
and (
  f.radius is null or f.lon is null or f.lat is null
  or (f.radius >= haversine(f.lat, f.lon, @currentLat, @currentLon)*6373)
);
```

- e. 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.

```
with
given_note as (
select * from Note natural join
Schedule natural join Location natural left outer join Attach where nid=@nid),
unfiltered_user as (
select u.uid, u.state, last_lat, last_lon from User u, given_note n
where (n.uid=u.uid
      or (visibleGroup=1 and u.uid in (select f.friendUid from Friendship f where f.uid=n.uid))
      or visibleGroup=2)
and (
  startTime is null or endTime is null
  or Time(@currentTime) between startTime and endTime
)
and (
  fromDate is null or endDate is null or repetition is null
  or (Date(@currentTime) between fromDate and endDate
  and datediff(Date(@currentTime),fromDate) % repetition=0)
)
and (
  visibleRadius is null or lon is null or lat is null
  or (visibleRadius >= haversine(lat, lon, last_lat, last_lon)*6373)
)
)
```

```
select distinct f.uid from given_note n, `Filter` f join unfiltered_user u on f.uid=u.uid
where (f.state is null or f.state=u.state)
and (f.tname is null or f.tname=n.tname)
and (
  f.startTime is null or f.endTime is null
  or Time(@currentTime) between f.startTime and f.endTime
)
and (
  f.fromDate is null or f.endDate is null or f.repetition is null
  or (Date(@currentTime) between f.fromDate and f.endDate
  and datediff(Date(@currentTime),f.fromDate) % f.repetition=0)
)
and (
  f.radius is null or f.lon is null or f.lat is null
  or (f.radius >= haversine(f.lat, f.lon, last_lat, last_lon)*6373)
);
```

- f. 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.

```
alter table inter_result add fulltext(noteContent);
select nid from inter_result
where match(noteContent) against (@keyword);
```

### 3. Test

By setting the placeholder to different values, we test the queries above on our data set:

- a. Create a new user account, with username “Anna Baker”, password “Anna123”, login/email “AnnaBaker@nyu.edu”, assuming the default state is “at work”, the user longitude is -71.069793, latitude is 42.353280.

```
set @password = "Anna123";
set @email = "AnnaBaker@nyu.edu";
set @default_state = "at work";
set @fake_lon = -71.069793;
set @fake_lat = 42.353280;
```

uid	username	password	email	state	last_lon	last_lat
100001	Antonio Rod	Antonio123	AntonioRod@gmail.com	at work	-73.9776	40.6837
100002	Jerry Tong	Jerry123	JerryTong@gmail.com	just chilling	-74.0114	40.7065
100003	Brown Snow	Brown123	BrownSnow@hotmail.com	just chilling	-73.9997	40.7093
100004	Guru Singh	Guru123	GuruSingh@gmail.com	lunch break	-122.413	37.7875
100005	Ivy Yu	Ivy123	IvyYu@hotmail.com	at work	-71.0698	42.3533
100006	Jack Smith	Jack123	JackSmith@gmail.com	lunch break	-73.9786	40.7573
100007	Jane Jing	Jane123	JaneJing@gmail.com	just chilling	-79.6568	43.5878
100008	Tom White	Tom123	TomWhite@gmail.com	on vacation	-122.325	47.6066
100009	Jose Luis	Jose123	JoseLuis@gmail.com	just chilling	-99.1318	19.4165
100010	Feihong Huang	Feihong123	FeihongHuang@qq.com	just chilling	121.101	31.144
100011	Anna Baker	Anna123	AnnaBaker@nyu.edu	at work	-71.0698	42.3533

- b. User 100010 posts a new note, saying “They have really good brunch”, with tag “Lunch” and “Noodle”, visible to friends and allow to comment, setting every day between 11 a.m. to 1 p.m., from 2018 to 2019. The

location is “Tim Ho Wan” (48.731490, -37.989910), setting the radius 100 km.

```
set @uid = 100010;
set @tname1 = "lunch";
set @tname2 = "noodle";
set @startTime = "11:00:00";
set @endTime = "13:00:00";
set @repetition = 1;
set @fromDate = "2018-01-01";
set @endDate = "2019-01-01";
set @lname = "Tim Ho Wan";
set @lon = -37.989910;
set @lat = 48.731490;
set @visibleRadius = 100;
set @publishTime = current_timestamp();
set @noteContent = "They have really good brunch";
set @visibleGroup = 1;
set @allowComment = True;
```

Note & Attach:

nid	uid	lid	sid	noteContent	publishTime	visibleRadius	visibleGroup	allowCc
10001	100003	102	1004	Food is delicious!!!	2018-10-05 13:00:00	100	2	1
10002	100002	101	1003	MoMa is awesome	2018-06-08 15:05:32	100	1	1
10003	100004	104	1001	WTC	2018-05-14 16:43:54	100	0	1
10004	100001	105	1002	Had a good dinner	2017-08-02 19:54:23	100	2	1
10005	100009	103	1003	Delicious food and resonable price	2018-03-21 14:12:06	100	2	1
10006	100010	108	1006	Awesome view up here	2018-10-11 20:06:43	100	2	1
10007	100008	106	1007	Walgreens is packed now	2018-07-19 13:34:34	100	2	1
10008	100006	107	1008	Busy palce	2018-05-17 12:45:50	100	2	1
10009	100010	109	1010	They have really good brunch	2018-11-29 01:45:10	100	1	1

Schedule & Location & Tag:

sid	startTime	endTime	repetition	fromDate	endDate	lid	lname	lon	lat	tname
1001	12:00:00	20:00:00	1	2017-10-05	2019-10-05	101	MoMA	-73.977	40.761	Dinner
1002	11:00:00	12:00:00	1	2017-10-05	2019-11-05	102	Cafe China	-73.9821	40.7501	Lunch
1003	09:00:00	12:00:00	2	2017-10-05	2019-11-05	103	Taiwan Pork Chop House	-73.998	40.7142	McDonalds
1004	15:00:00	15:30:00	3	2017-10-05	2019-11-05	104	World Trade Center	-74.0134	40.7127	Michelin
1005	16:00:00	18:00:00	4	2017-10-05	2019-11-05	105	Wild Ginger Seattle	-122.337	47.6088	noodle
1006	19:00:00	19:45:00	5	2017-10-05	2019-11-05	106	Walgreens	-122.412	37.8015	NYC
1007	20:00:00	20:30:00	6	2017-10-05	2019-12-05	107	McDonalds	-99.1554	19.425	Sightseeing
1008	09:00:00	10:00:00	7	2017-10-05	2019-12-05	108	Dong Fang Ming Zhu Dian Shi Ta	121.495	31.2419	Walgreens
1009	08:00:00	08:30:00	8	2017-10-05	2019-12-05	109	Tim Ho Wan	-37.9899	48.7315	
1010	11:00:00	13:00:00	1	2018-01-01	2019-01-01					

c. Show all friends of user 100001.

```
friendUid
100002
100003

set @uid=100011;
```

d. For user#100003, who is standing at (40.7093, -73.9997) at 2018-11-05 11:20:00, with state “just chilling”, output all notes the user can see.

```
set @uid=100003;  
set @currentTime="2018-11-05 11:20:00";  
set @currentLon=-73.977;  
set @currentLat=40.761;  
set @state="just chilling";
```

nid
10005

- e. For note#100002, output all users that should see it at 2018-11-05 11:20:00.

```
set @nid = 10005;  
set @currentTime="2018-11-05 11:20:00";
```

uid
100002
100003

- f. Assume the intermediate result is in Table `inter\_result` like below:

nid	uid	lid	sid	noteContent	publishTime	visibleRadius	visibleGroup	allowComment
10001	100003	102	1004	Food is delicious!!!	2018-10-05 13:00:00	100	2	1
10004	100001	105	1002	Had a good dinner	2017-08-02 19:54:23	100	2	1
10005	100009	103	1003	Delicious food and resonable price	2018-03-21 14:12:06	100	2	1
10006	100010	108	1006	Awesome view up here	2018-10-11 20:06:43	100	2	1

Further filter the content with keyword “food” and “view”.

```
set @keyword = "food view";
```

nid
10006
10001
10005

## Robustness

- SQL injection prevention

Since the project utilizes Django framework and the framework itself is immune to SQL injection, SQL injection is hard. The Django treats queries as data, NOT commands. The project doesn't use any string concatenation (e.g. `raw()` SQL queries). So, the project is immune to SQL injection.

---

*By using Django's query sets, the resulting SQL will be properly escaped by the underlying database driver. However, Django also gives developers power to write raw queries or execute custom SQL. These capabilities should be used sparingly, and you should always be careful to properly escape any parameters that the user can control. In addition, users should exercise caution when using `extra()`.*

*Django Docs*

---

- Cross-Site scripting protection

The project has auto-escaping enabled, so it is immune to most XSS. However, it is not 100% secure to XSS.



- All attributes were quoted where dynamic data is inserted.
- the project doesn't insert data into attributes in URLs.
- No data is inserted into comments.
- The project doesn't use mark\_safe for any data, so no data is treated as safe.
- Client side XSS
  - The main HTML document is static.
  - No untrusted data transmitted in same HTML.
  - No untrusted data transmitted from server to client.
  - DOM based XSS is not possible since the project doesn't use the API

---

*Using Django templates protects you against the majority of XSS attacks. However, it is important to understand what protections it provides and its limitations.*

*Django templates escape specific characters which are particularly dangerous to HTML. While this protects users from most malicious input, it is not entirely foolproof. For example, it will not protect the following:*

```
<style class={{ var }}>...</style>
```

*If var is set to 'class1 onmouseover=javascript:func()', this can result in unauthorized JavaScript execution, depending on how the browser renders imperfect HTML. (Quoting the attribute value would fix this case.)*

---

- Concurrency and scalability

The system is robust in terms of concurrency; the scalability of the system is impressive, and the system can server very large number of users at the same time. There are some famous sites built on Django, including Instagram, Pinterest, Bitbucket, etc.

## Revised designs

- New functionalities

For users' convenience, the following functionalities were added into the system:

- Manually setting location and time of users. A webpage is provided, and the link button is provided in user interface. This is for the convenience of giving demo. However, users may want to see notes in some area and between some time span that he/she is interested in for

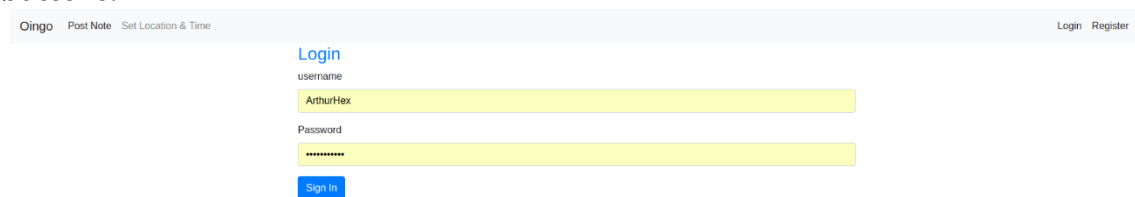


future visits. Google Maps API is provided for users to choose locations.

- Showing profile of the user logged-in.
  - Deleting a friend (unfriend). It can be achieved by users when he/she opens the friend list. The unfriend button is next every friend of her/his.
  - Showing notes written by the logged-in user. The user can click on the button and be redirected to another webpage showing all his/her notes in detail.
  - Editing posted notes. Users can edit posted notes created by them. For example, the user wants to edit visible radius when he/she notices the note has not been commented by many users.
  - Removing notes. Users can delete notes created by them if they change their mind later.
  - Users can see their own filters and edit/remove them if they want.
- Modified functionalities
    - Friendship management is changed. We added a new attribute called `is_requested` in the Friendship entity. The attribute indicates that if the user has requested to be friend of another user. The attribute is used for rendering friend request on user interface.

## User interface

The system is easy to use because main functionalities are obviously shown as buttons.



The screenshot shows a web interface with a navigation bar at the top containing links: 'Oingo', 'Post Note', and 'Set Location & Time'. On the right side of the navigation bar are links for 'Login' and 'Register'. Below the navigation bar is a 'Login' section. It includes a label 'username' above a text input field containing 'ArthurHex'. Below that is a label 'Password' above a password input field filled with dots. At the bottom of the login section is a blue button labeled 'Sign In'.

Guanwei Zhang – gz667, Chris Fan – cf2240  
CS 6083 PROJECT 2

Oingo Post Note Set Location & Time Login Register

Register

username

Email address

Password

Re-password

Sign Up

Oingo Post Note Set Location & Time welcome, ArthurHex

Create Note:

content:

Tags:(separate with ";")

start time:

end time:

repetition: Every day

from date: mm/dd/yyyy

to date: mm/dd/yyyy

location name:

lat:

lon:

Click to set coordinates.

visible radius(km):

visible group: self only

allow comment:

save back

Oingo Post Note Set Location & Time welcome, ArthurHex

Create Note:

content:

Tags:(separate with ";")

start time:

end time:

repetition: Every day

from date: mm/dd/yyyy

to date: mm/dd/yyyy

location name:

lat:

lon:

Click to set coordinates.

visible radius(km):

visible group: self only

allow comment:

save back

Profile

Friends

Filter

Own Notes

Logout