# PROJECT REPORT
# PART-I

SANDEEP NANDIMANDALAM - sn2610
ANVITA MARLA - am9435

## INDEX

## <u>OVERVIEW :</u>

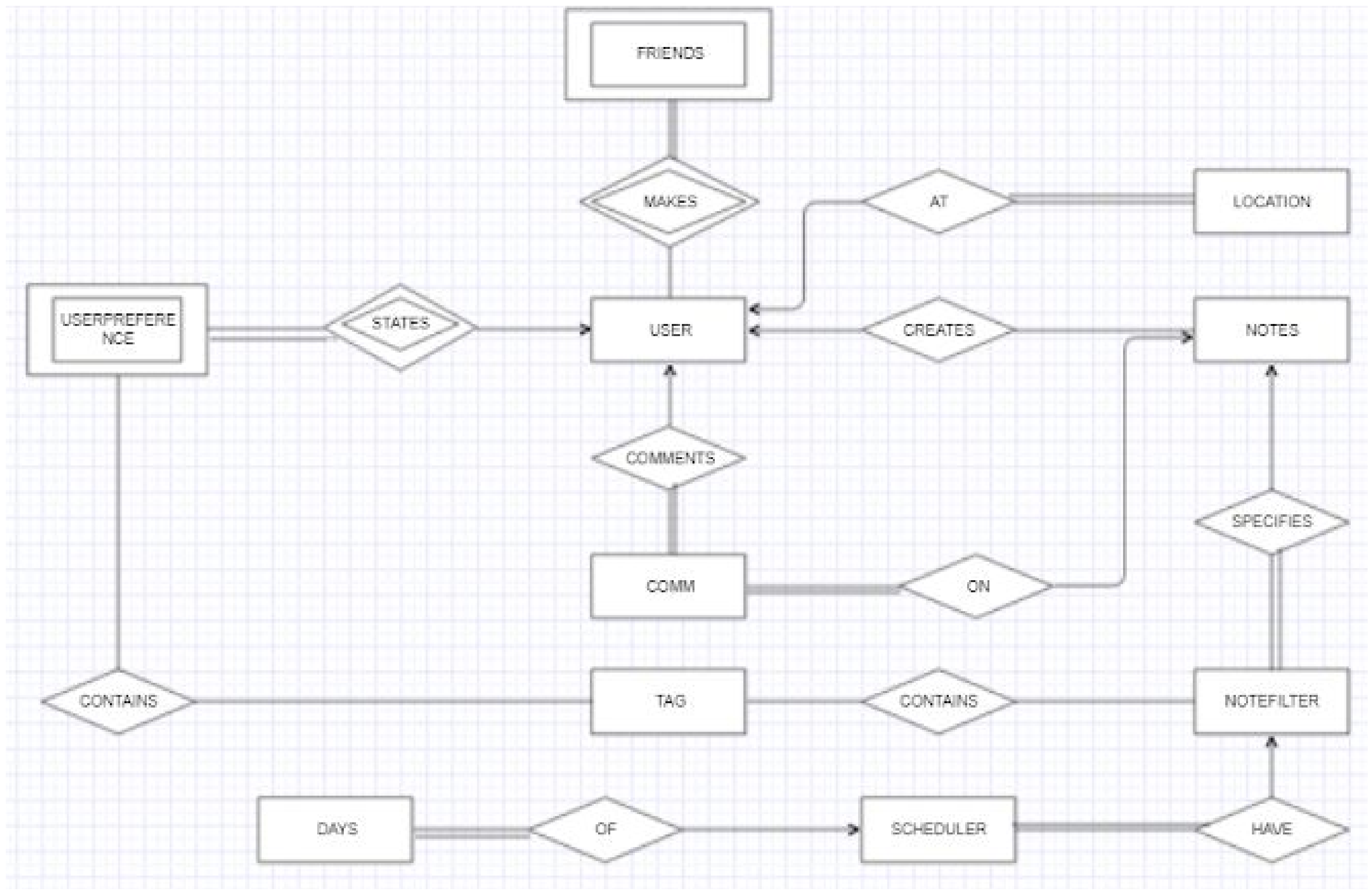
The purpose of this report is to provide a database design for Oingo. The database design is described using an ER diagram and a relational schema based on the proposed ER diagram.

Oingo is a startup that wants to create a platform for users to be able to publish short notes on useful information like a restaurant serving some special food or some special event going on at some place, which can be viewed/received by other users depending on their state and location.

Users can register on oingo and set different states, each state has some preferences attached to it, as to what kind of notes he wants to see. The user can switch between these states. Users can also publish notes. The user who publishes notes, also attributes some tags to the notes and specifies the area in which a particular note has to be visible and a schedule as to when a note must be shown to interested users. Now based on these filters set on a particular note, we match users (considering their current state) who would be delighted to view the note. We can also find all the notes that fit the preferences of a particular user. Users are also allowed to comment on notes (if comments are enabled for that note).

This report puts forth an ER model and a relational schema. A brief description as to how our design handles the requirements and some limitations of our design has been given. A few use cases of the idea have been shown in the form of SQL queries.

## <u>ER DIAGRAM :</u>



**Entities -**

user(<u>userid</u>, firstname, lastname, uemail, password, userstate)
notes(<u>noteid</u>, notedesc, userid, comments)
friends(status)
notefilter(<u>nid</u>, lat, long, rad)
userpref(<u>state</u>, stime, etime, radius )
location(<u>lid</u>, latitude, longitude, ts)
tag(<u>tagid</u>, tagname)
scheduler(<u>sid</u>, fromdate, todate)
days(<u>dayid</u>, fromtime, totime, daay)
comment (<u>cid</u>, noteid, userid, commentdesc, timestamps)

**Design -**

One user can have many friends. Friends is a weak entity as it need users to identify which two users are friends or not. User can have states like ‘lunch’, ‘dinner’, etc and set some preferences for each state like what kind of notes he wants to see and the start time and endtime of that state and the area of his interest when he is in that state. Userpreference is a weak entity as it has no identity of its own without a user associated with it. Userpreference also has tags associated with it, that is, every state specified by the user also contains tags which are used to filter notes based on the present state of the user. User has a location which is updated every 5 minutes or whenever he makes some changes or does some activity like changing his state or searching for notes with a particular tag, etc. Every location is associated with only one user. Notes are written by users, and one note can be posted only by one user. Every note has notefilters which specifies the area in which the notes can be seen and notefilters also contain tags. The tags have been associated with notefilters rather than notes as a particular note might have multiple notefilters depending on if the note has to be visible to users at multiple locations and the user writing this note wants to have different tags for these different locations. One notefilter can be associated only with one note. The scheduler is the schedule as to when the notes has to be visible to prospective users. This schedule is associated with a notefilter as one note can have multiple notefilters and each notefilter can have a different schedule depending on the location. Each notefilter itself can have multiple schedules, where in he can specify periods of dates in which that particular note has to be displayed to users. One schedule must be associated only with one notefilter. Each schedule has a startdate and an enddate and to accommodate requirements like “Everyday in a particular date range” or “Every Monday and Tuesday in particular date range” or “Every wednesday from 2pm to 5pm in a date range” or “2pm to 5pm on Mondays and 10am to 1pm on Wednesday in a particular date range” we have created another entity called days which stores the time range and the day on which a particular note has to be considered. Days must be associated with only one schedule. Users can comment on a note. One comment can be made by only one user on one particular note.

Here if a note has to be visible to interested users “Every Monday on the first week and third week of January” or “Biweekly every month”, there is no direct way this can be done in our design, the user who is writing the notes has to separately enter the date range of each week and then specify the days and time accordingly. Considering these cases would be very limited, we have decided to not accommodate these cases in the thought that handling these cases would complicate the design.

## Converting to Relational Schema:

The user and tag table are individual entities and contain no foreign keys.
The friends table has to have two columns as userid, fuserid which are combinedly the primary key of this table and also are point to userid in user. Here we ensure userid < fuserid, so as to avoid duplicate entries.
The userpreference table will have userid, state as its primary key. This also ensures that a user can add multiple states and also ensures that one user cannot have same states with different preferences as this would create ambiguity.
The userid in the location table which points to the userid in the user table to make sure one location points to only one user.
The notes table will have userid to identify which user created the notes, and userid points to the userid in user table. This ensures one note can be created only by one user.
The notefilter table contains noteid, as it can only be associated with one note according to our design.
The utag is associated with userid, state in userpreference table and tagid in tag table. This table models the relation between userpreference and tag table wherein one userpreference can contain multiple tags.
The ntag is associated with nid in notefilter table and tagid in tag table. This table models the relation between notefilter and tag table wherein one notefilter can contain multiple tags.
The scheduler table maintains the schedule for a particular notefilter. So we put nid in the scheduler table which also ensures that one schedule points to only one notefilter.
The days table has sid that references sid in scheduler and also ensures one day record can be associated only with one schedule

user(userid, firstname, lastname, uemail, password, userstate)
notes(noteid, notedesc, userid,comments)
friends(userid, fuserid, status,fromid)
notefilter(nid, noteid, lat, long, rad)
userpreference(userid, state, stime, etime, radius )
location(lid, userid, latitude, longitude, ts)
tag(tagid, tagname)
utag(userid, tagid, state)
ntag(nid, tagid)

scheduler(<u>sid</u>, nid, fromdate, todate)
days(<u>dayid</u>, sid, fromtime, totime, daay)
comm (<u>cid</u>, noteid, userid, commentdesc, timestamps)

**Primary Keys:**

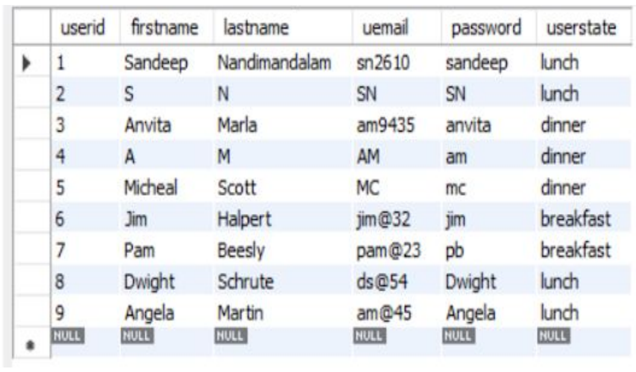
user - Primary Key (userid)
notes - Primary Key (noteid)
friend - Primay Key (userid,fuserid)
notefilter - Primary Key (nid)
userpreference - Primary Key (userid, state)
location - Primary Key (lid,userid)
tag - Primary Key (tagid)
utag - Primary Key (userid,state)
ntag - Primary Key (nid, tagid)
scheduler - Primary Key (sid)
days - Primary Key (dayid)
comm - Primary Key (cid)

**Foreign keys :**

notes (userid) REFERENCES user(userid);
friends (userid) REFERENCES user(userid);
friends (fuserid) REFERENCES user(userid);
friends (fromid) REFERENCES user(userid);
userpref (userid) REFERENCES user(userid);
location (userid) REFERENCES user(userid);
notefilter (noteid) REFERENCES notes(noteid);
utag(tagid) REFERENCES tag(tagid);
utag (userid,state) REFERENCES userpref(userid,state);
ntag (tagid) REFERENCES tag(tagid);
ntag (nid) REFERENCES notefilter(nid);
scheduler (nid) REFERENCES notefilter(nid);
days (sid) REFERENCES scheduler(sid);
comm (noteid) REFERENCES notes(noteid)
comm (userid) REFERENCES user(userid)

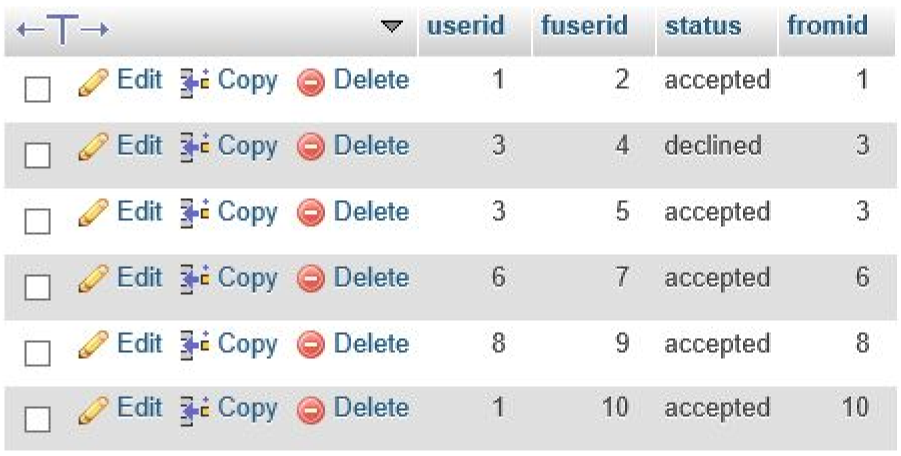
## **Tables:**

user table :



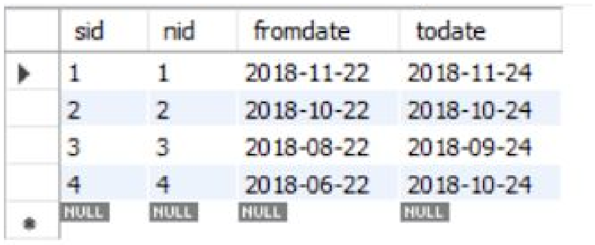
| userid | firstname | lastname | uemail | password | userstate |
|---|---|---|---|---|---|
| 1 | Sandeep | Nandimandalam | sn2610 | sandeep | lunch |
| 2 | S | N | SN | SN | lunch |
| 3 | Anvita | Marla | am9435 | anvita | dinner |
| 4 | A | M | AM | am | dinner |
| 5 | Micheal | Scott | MC | mc | dinner |
| 6 | Jim | Halpert | jim@32 | jim | breakfast |
| 7 | Pam | Beesly | pam@23 | pb | breakfast |
| 8 | Dwight | Schrute | ds@54 | Dwight | lunch |
| 9 | Angela | Martin | am@45 | Angela | lunch |
| NULL | NULL | NULL | NULL | NULL | NULL |

friends table:



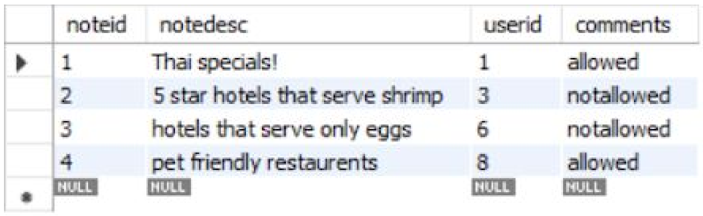
| | userid | fuserid | status | fromid |
|---|---|---|---|---|
| Edit Copy Delete | 1 | 2 | accepted | 1 |
| Edit Copy Delete | 3 | 4 | declined | 3 |
| Edit Copy Delete | 3 | 5 | accepted | 3 |
| Edit Copy Delete | 6 | 7 | accepted | 6 |
| Edit Copy Delete | 8 | 9 | accepted | 8 |
| Edit Copy Delete | 1 | 10 | accepted | 10 |

scheduler table:



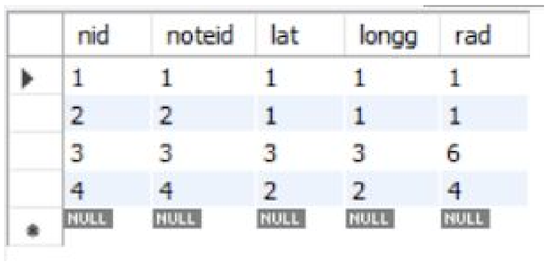
| sid | nid | fromdate | todate |
|---|---|---|---|
| 1 | 1 | 2018-11-22 | 2018-11-24 |
| 2 | 2 | 2018-10-22 | 2018-10-24 |
| 3 | 3 | 2018-08-22 | 2018-09-24 |
| 4 | 4 | 2018-06-22 | 2018-10-24 |
| NULL | NULL | NULL | NULL |

notes table:



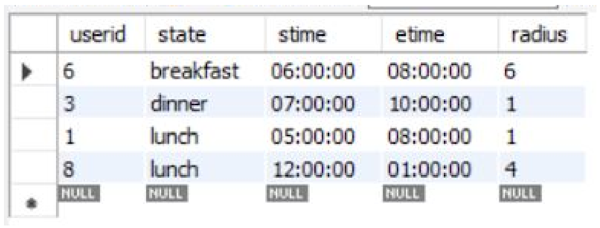
| noteid | notedesc | userid | comments |
|---|---|---|---|
| 1 | Thai specials! | 1 | allowed |
| 2 | 5 star hotels that serve shrimp | 3 | notallowed |
| 3 | hotels that serve only eggs | 6 | notallowed |
| 4 | pet friendly restaurents | 8 | allowed |
| NULL | NULL | NULL | NULL |

notefilter table:



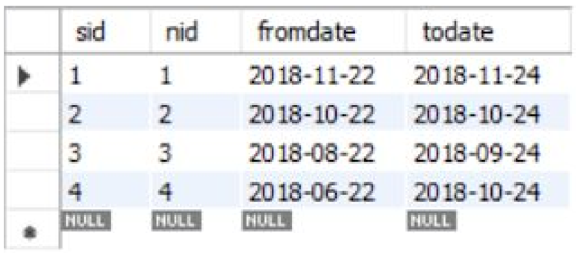
| nid | noteid | lat | longg | rad |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 1 | 1 |
| 3 | 3 | 3 | 3 | 6 |
| 4 | 4 | 2 | 2 | 4 |
| NULL | NULL | NULL | NULL | NULL |

userpref table:



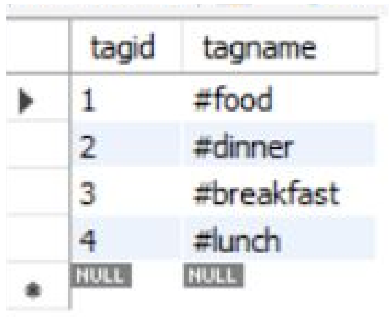
| userid | state | stime | etime | radius |
|---|---|---|---|---|
| 6 | breakfast | 06:00:00 | 08:00:00 | 6 |
| 3 | dinner | 07:00:00 | 10:00:00 | 1 |
| 1 | lunch | 05:00:00 | 08:00:00 | 1 |
| 8 | lunch | 12:00:00 | 01:00:00 | 4 |
| NULL | NULL | NULL | NULL | NULL |

scheduler table:



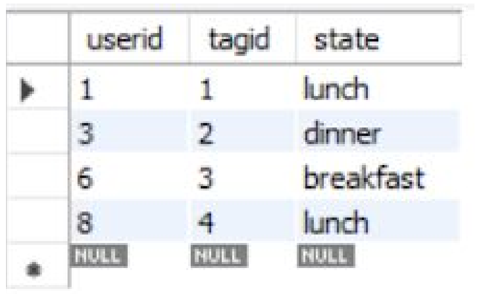
| sid | nid | fromdate | todate |
|---|---|---|---|
| 1 | 1 | 2018-11-22 | 2018-11-24 |
| 2 | 2 | 2018-10-22 | 2018-10-24 |
| 3 | 3 | 2018-08-22 | 2018-09-24 |
| 4 | 4 | 2018-06-22 | 2018-10-24 |
| NULL | NULL | NULL | NULL |

tag table:



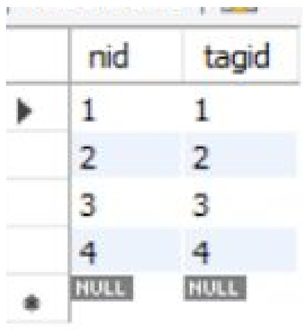
| tagid | tagname |
|---|---|
| 1 | #food |
| 2 | #dinner |
| 3 | #breakfast |
| 4 | #lunch |
| NULL | NULL |

utag table:



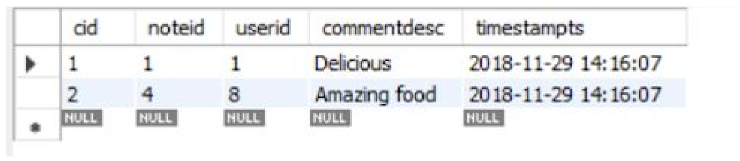
| userid | tagid | state |
|---|---|---|
| 1 | 1 | lunch |
| 3 | 2 | dinner |
| 6 | 3 | breakfast |
| 8 | 4 | lunch |
| NULL | NULL | NULL |

ntag table:



| nid | tagid |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| NULL | NULL |

comm table:



| cid | noteid | userid | commentdesc | timestampts |
|---|---|---|---|---|
| 1 | 1 | 1 | Delicious | 2018-11-29 14:16:07 |
| 2 | 4 | 8 | Amazing food | 2018-11-29 14:16:07 |
| NULL | NULL | NULL | NULL | NULL |

## Description:

Schedule:
We model schedules by maintaining a scheduler table/entity identified by sid where it also stores the information regarding the notefilter, fromdate and todate. Fromdate and todate is the range over which that particular notefilter is active.

Location:
Similarly, we model location by maintaining a location entity/table identified by lid where it stores information regarding the user's current location (latitude and longitude) and time.

Areas:
Additionally, we also keep track of area by the use of the radius atttribute in userpreference table which is the radius within which the user tags are active.

## Constraints:

In the friends relation, Check (userid < fuserid).

In the friends relation, Check ( status in (‘accepted’, ‘declined’, ‘ignored’))

In the days relation, Check ( daay in (‘Everyday’, ‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’, ‘Friday’, Saturday’, ‘Sunday’) )

## Queries:

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

   **Query:**

   **INSERT INTO** \`user\` **VALUES** (1, 'Sandeep', 'Nandimandalam', 'sn2610','sandeep', 'lunch');

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

   **Queries:**

   **INSERT INTO** \`user\` **VALUES** (1, 'Sandeep', 'Nandimandalam', 'sn2610','sandeep', 'lunch');

**INSERT INTO** `user` **VALUES** (2, 'S', 'N', 'SN','SN', 'lunch');
**INSERT INTO** `friends` **VALUES** (1,2, 'accepted');

**INSERT INTO** notes(noteid,notedesc,userid,comments) **VALUES** (1,"Thai specials!",1,"allowed");
**INSERT INTO** notefilter(nid,noteid,lat,longg,rad) **VALUES** (1,1,1,1,1);
**INSERT INTO** userpref(userid,state,stime,etime,radius) **VALUES** (1,'lunch','5:00','8:00',1);
**INSERT INTO** tag(tagid,tagname) **VALUES** (1,"#food");
**INSERT INTO** utag(userid,tagid,state) **VALUES** (1,1,'lunch');
**INSERT INTO** ntag(nid,tagid) **VALUES** (1,1);
**INSERT INTO** location(lid,userid,latitude,longitude,ts) **VALUES** (1,1,0.5,0.5,now());
**INSERT INTO** scheduler(sid,nid,fromdate,todate) **VALUES** (1,1,'2018-11-22','2018-11-24');
**INSERT INTO** days(dayid,sid,fromtime,totime,daay) **VALUES** (1,1,'05:00:00', '08:00:00', 'everyday');
**INSERT INTO** comm(cid,userid,noteid,commentdesc,timestampts) **VALUES** (1,1,1,"Delicious", now());

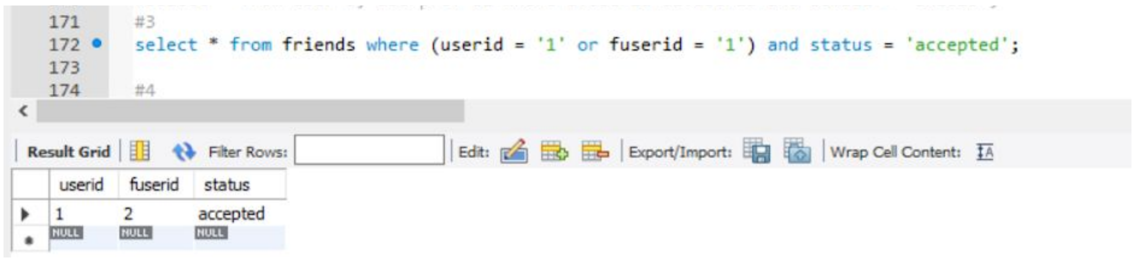
3. For a given user, list all friends.

**Query :**

If the given userid is 1,

**SELECT** * **FROM** friends
**WHERE** (userid = '1' or fuserid = '1') and status = 'accepted';

**Output :**



```
#3
select * from friends where (userid = '1' or fuserid = '1') and status = 'accepted';

#4
```

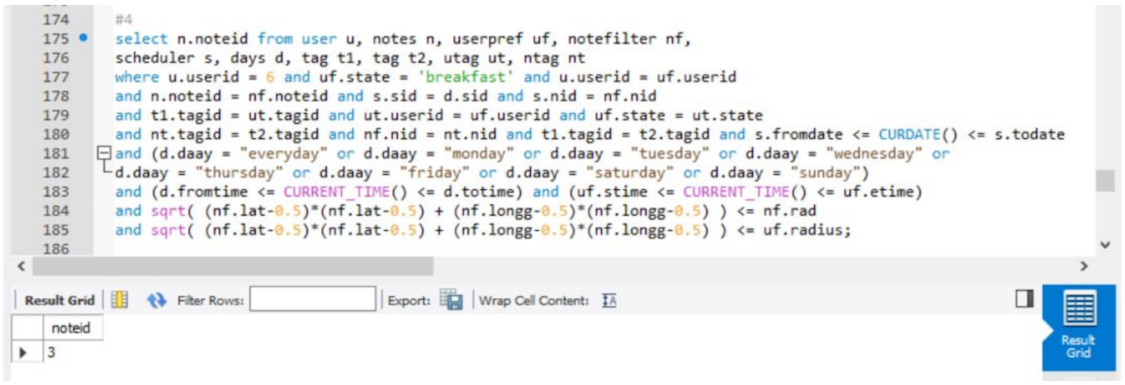
```
userid  fuserid  status
1       2        accepted
NULL    NULL     NULL
```

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.

**Query:**
If the given userid is 6 whose state is breakfast,

**SELECT** n.noteid
**FROM** user u, notes n, userpref uf, notefilter nf, scheduler s, days d, tag t1, tag t2, utag ut, ntag nt
**WHERE** u.userid = 6 and uf.state = 'breakfast' and u.userid = uf.userid and n.noteid = nf.noteid and s.sid = d.sid and s.nid = nf.nid and t1.tagid = ut.tagid and ut.userid = uf.userid and uf.state = ut.state and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid and s.fromdate <= CURDATE() <= s.todate and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday") and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime) and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= nf.rad and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= uf.radius;

**Output :**



```
#4
select n.noteid from user u, notes n, userpref uf, notefilter nf,
scheduler s, days d, tag t1, tag t2, utag ut, ntag nt
where u.userid = 6 and uf.state = 'breakfast' and u.userid = uf.userid
and n.noteid = nf.noteid and s.sid = d.sid and s.nid = nf.nid
and t1.tagid = ut.tagid and ut.userid = uf.userid and uf.state = ut.state
and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid and s.fromdate <= CURDATE() <= s.todate
and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or
d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday")
and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime)
and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= nf.rad
and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= uf.radius;
```

```
noteid
3
```

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.

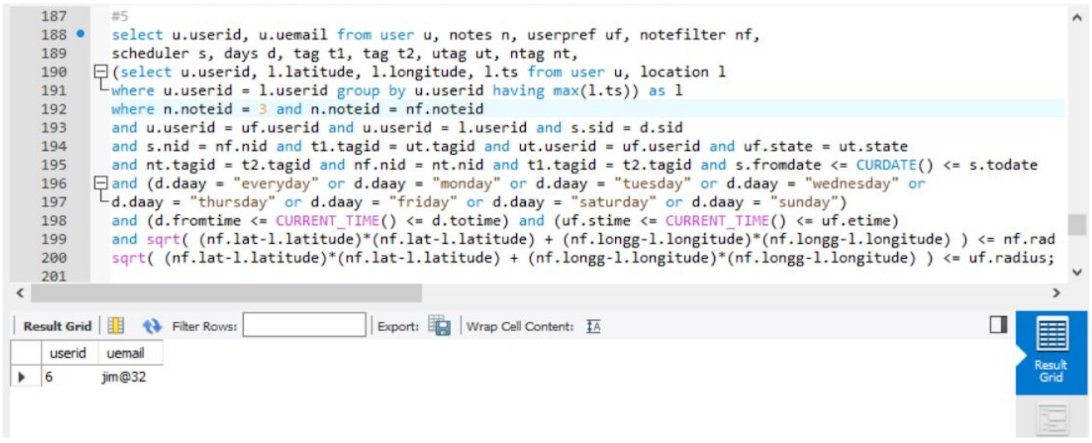
**Query:**

If the given noteid is 3,

**SELECT** u.userid, u.uemail
**FROM** user u, notes n, userpref uf, notefilter nf, scheduler s, days d, tag t1, tag t2, utag ut, ntag nt, (select u.userid, l.latitude, l.longitude, l.ts from user u, location l where

u.userid = l.userid group by u.userid having max(l.ts)) as l
**WHERE** n.noteid = 3 and n.noteid = nf.noteid and u.userid = uf.userid and u.userid = l.userid and s.sid = d.sid and s.nid = nf.nid and t1.tagid = ut.tagid and ut.userid = uf.userid and uf.state = ut.state and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid and s.fromdate <= CURDATE() <= s.todate and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday") and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime) and sqrt( (nf.lat-l.latitude)*(nf.lat-l.latitude) + (nf.longg-l.longitude)*(nf.longg-l.longitude) ) <= nf.rad and sqrt( (nf.lat-l.latitude)*(nf.lat-l.latitude) + ( nf.longg-l.longitude) * (nf.longg-l.longitude)) <= uf.radius;

**Output :**



```
#5
select u.userid, u.uemail from user u, notes n, userpref uf, notefilter nf,
scheduler s, days d, tag t1, tag t2, utag ut, ntag nt,
(select u.userid, l.latitude, l.longitude, l.ts from user u, location l
where u.userid = l.userid group by u.userid having max(l.ts)) as l
where n.noteid = 3 and n.noteid = nf.noteid
and u.userid = uf.userid and u.userid = l.userid and s.sid = d.sid
and s.nid = nf.nid and t1.tagid = ut.tagid and ut.userid = uf.userid and uf.state = ut.state
and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid and s.fromdate <= CURDATE() <= s.todate
and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or
d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday")
and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime)
and sqrt( (nf.lat-l.latitude)*(nf.lat-l.latitude) + (nf.longg-l.longitude)*(nf.longg-l.longitude) ) <= nf.rad
sqrt( (nf.lat-l.latitude)*(nf.lat-l.latitude) + (nf.longg-l.longitude)*(nf.longg-l.longitude) ) <= uf.radius;
```

| userid | uemail |
|---|---|
| 6 | jim@32 |

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.

**Query:**

If the given userid is 8,

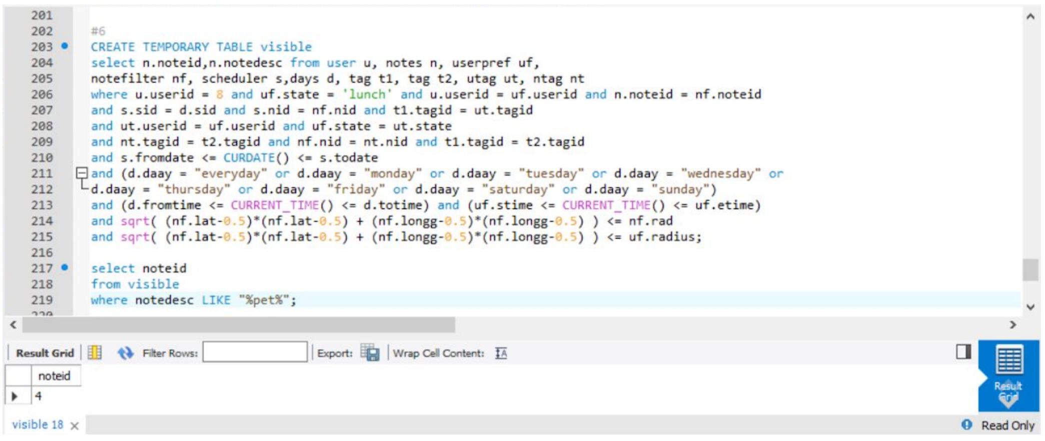
**CREATE TEMPORARY TABLE** visible
**SELECT** noteid, notedesc

**FROM** user u, notes n, userpref uf, notefilter nf, scheduler s, days d, tag t1, tag t2, utag ut, ntag nt
**WHERE** u.userid = 8 and uf.state = 'lunch' and u.userid = uf.userid and n.noteid = nf.noteid and s.sid = d.sid and s.nid = nf.nid and t1.tagid = ut.tagid and ut.userid = uf.userid and uf.state = ut.state and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid and s.fromdate <= CURDATE() <= s.todate and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday") and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime) and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= nf.rad and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= uf.radius

If the keyword is “pet”

**SELECT** noteid
**FROM** visible
**WHERE** notedesc LIKE “%pet%”

**Output :**



```
#6
CREATE TEMPORARY TABLE visible
select n.noteid,n.notedesc from user u, notes n, userpref uf,
notefilter nf, scheduler s,days d, tag t1, tag t2, utag ut, ntag nt
where u.userid = 8 and uf.state = 'lunch' and u.userid = uf.userid and n.noteid = nf.noteid
and s.sid = d.sid and s.nid = nf.nid and t1.tagid = ut.tagid
and ut.userid = uf.userid and uf.state = ut.state
and nt.tagid = t2.tagid and nf.nid = nt.nid and t1.tagid = t2.tagid
and s.fromdate <= CURDATE() <= s.todate
and (d.daay = "everyday" or d.daay = "monday" or d.daay = "tuesday" or d.daay = "wednesday" or
d.daay = "thursday" or d.daay = "friday" or d.daay = "saturday" or d.daay = "sunday")
and (d.fromtime <= CURRENT_TIME() <= d.totime) and (uf.stime <= CURRENT_TIME() <= uf.etime)
and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= nf.rad
and sqrt( (nf.lat-0.5)*(nf.lat-0.5) + (nf.longg-0.5)*(nf.longg-0.5) ) <= uf.radius;

select noteid
from visible
where notedesc LIKE "%pet%";
```

| noteid |
|---|
| 4 |

## <u>Description</u>:

Oingo provides services only for authenticated users. Hence, users are first needed to register before logging in. During registration we ask the user for a default state based on which we can primarily filter his/her preferred notes. Once the user logs in, it'll redirect him/her to a homepage where all his/her personalized notes will be displayed. He/she can always log out at anytime by making use of the 'logout' tab on the navigation bar.

The homepage provides flexibility to the user where he/she can choose from various options. Based on the default state set by the user at the time of registration, he/she can view notes by clicking on 'Notes for you' tab. He/she can always change his/her state by changing the state under the 'Settings' tab. A user can also view all notes by clicking on 'All Notes' tab which displays all notes available. One can always view their own created notes under the 'Your Notes' tab. A user can always change his/her preferences by going to the 'Settings' tab.

Oingo also provides facility to add friends by sending them a friend request. A user can do this by going to the settings dropdown and selecting add friend. After going to add friend the user can search for friends using their name and send a friend request. He/she can also, at any time, remove a particular friend by selecting 'Remove friend' option under 'Settings' tab. A user is also provided a search bar in the 'Notes For You', 'All Notes' and 'Your Notes' tabs. This search can be done based on tags inputted by the user or the description of the notes he/she is searching for.

The 'Settings' tab has many modules. A user can add new notes, add new states, change current state and add/remove friends. He/she can change his preferences by changing these values.

## <u>Design Drawbacks:</u>

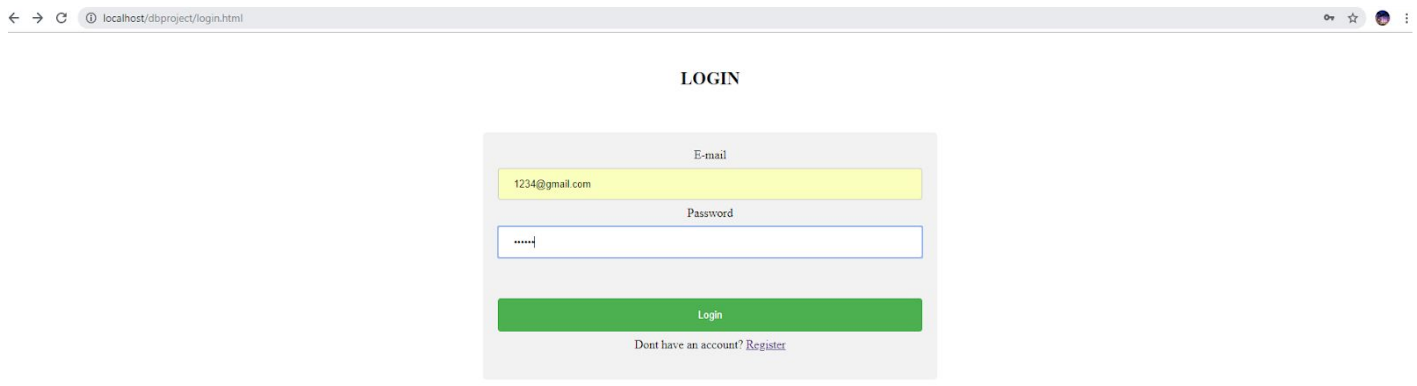
We are not populating the Location table, this is because we are getting the current location of the user from the browser if he allows it, or we have decided not to show notes if the user denies location access. The user can still use other tabs and features.

We are also not supporting schedules like "Every Monday Biweekly" , etc, since this would complicate our design.

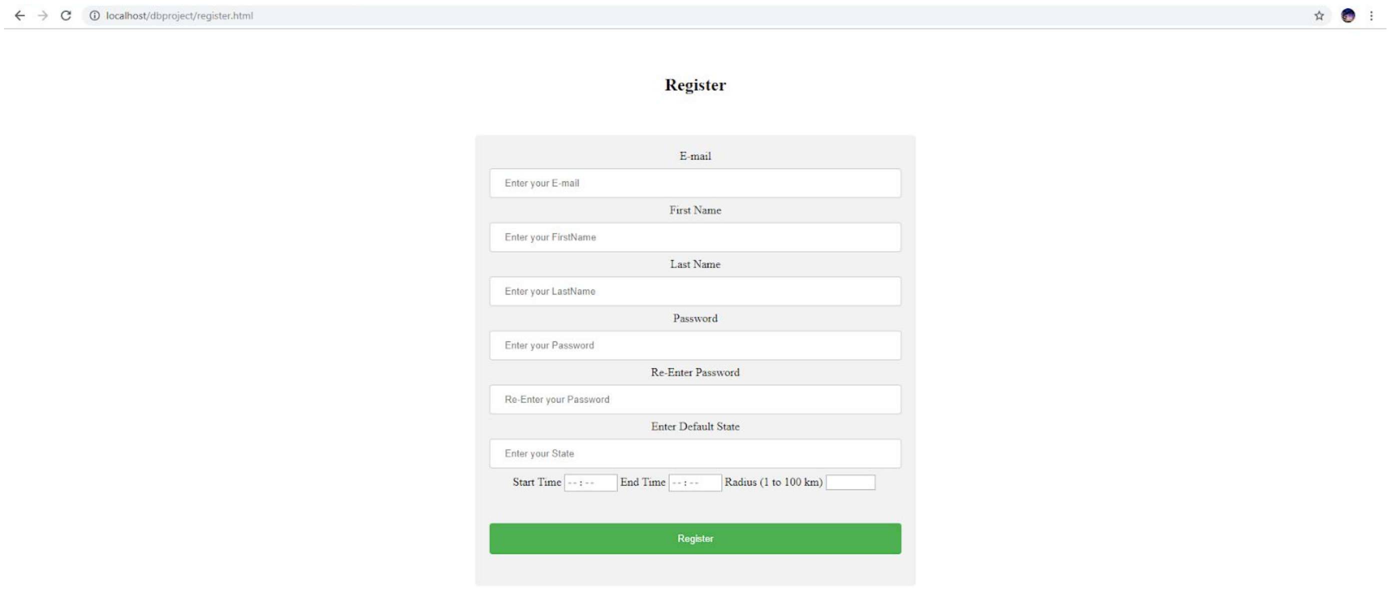
We are taking in coordinates from the map when a user wants to add notes and not the name of the place as this would otherwise complicate our design. This also wouldn't make a big difference.

## Screens:

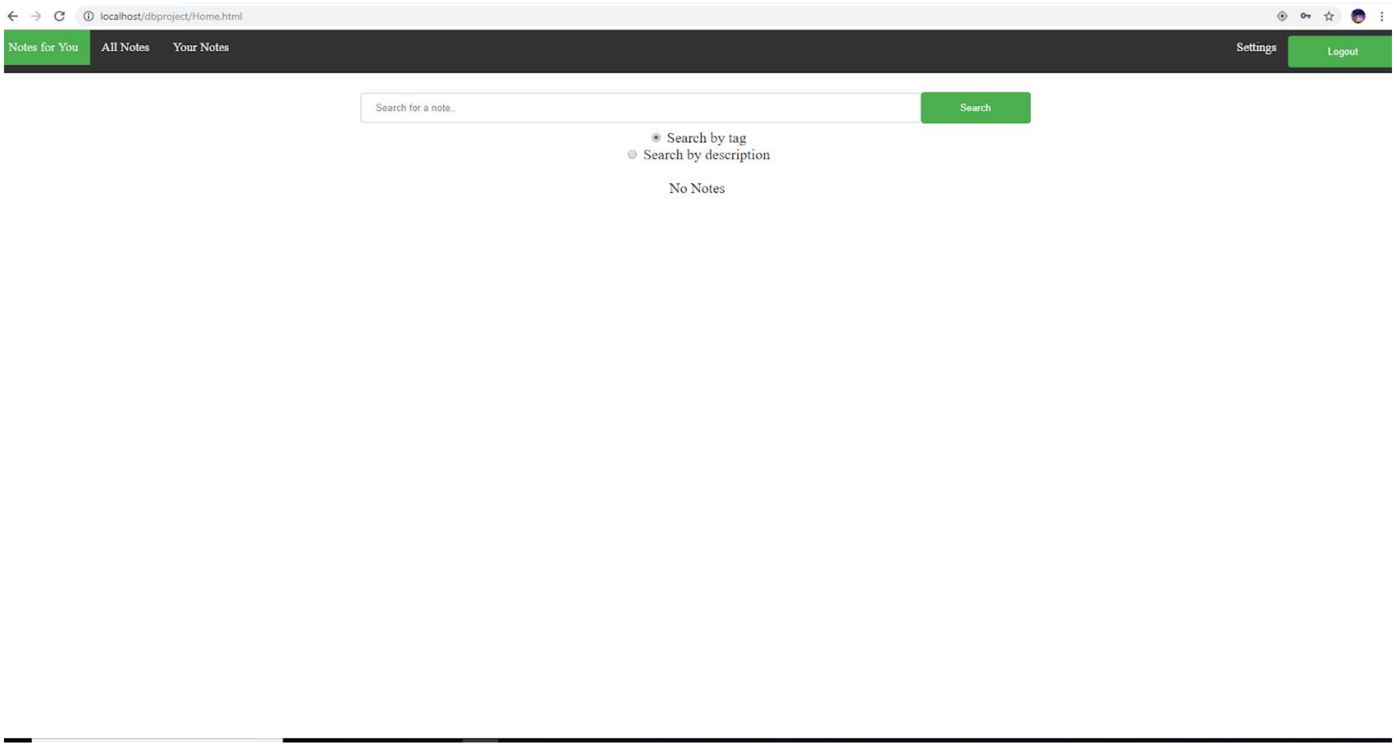
1. Login Page, option to register via register link.


localhost/dbproject/login.html
LOGIN
E-mail
1234@gmail.com
Password
Login
Dont have an account? Register

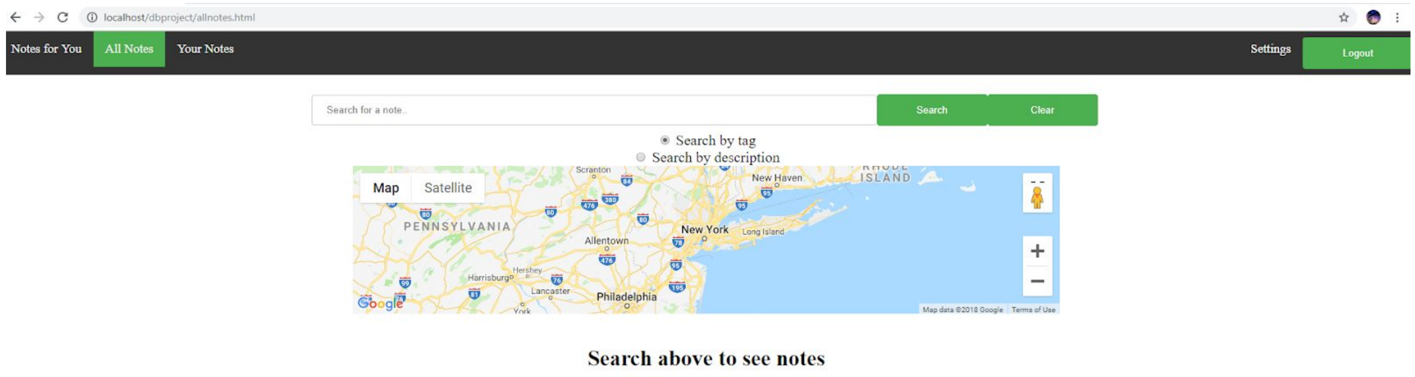
2. Register Page



3. Notes For You Tab, with a facility to search notes on tag and description

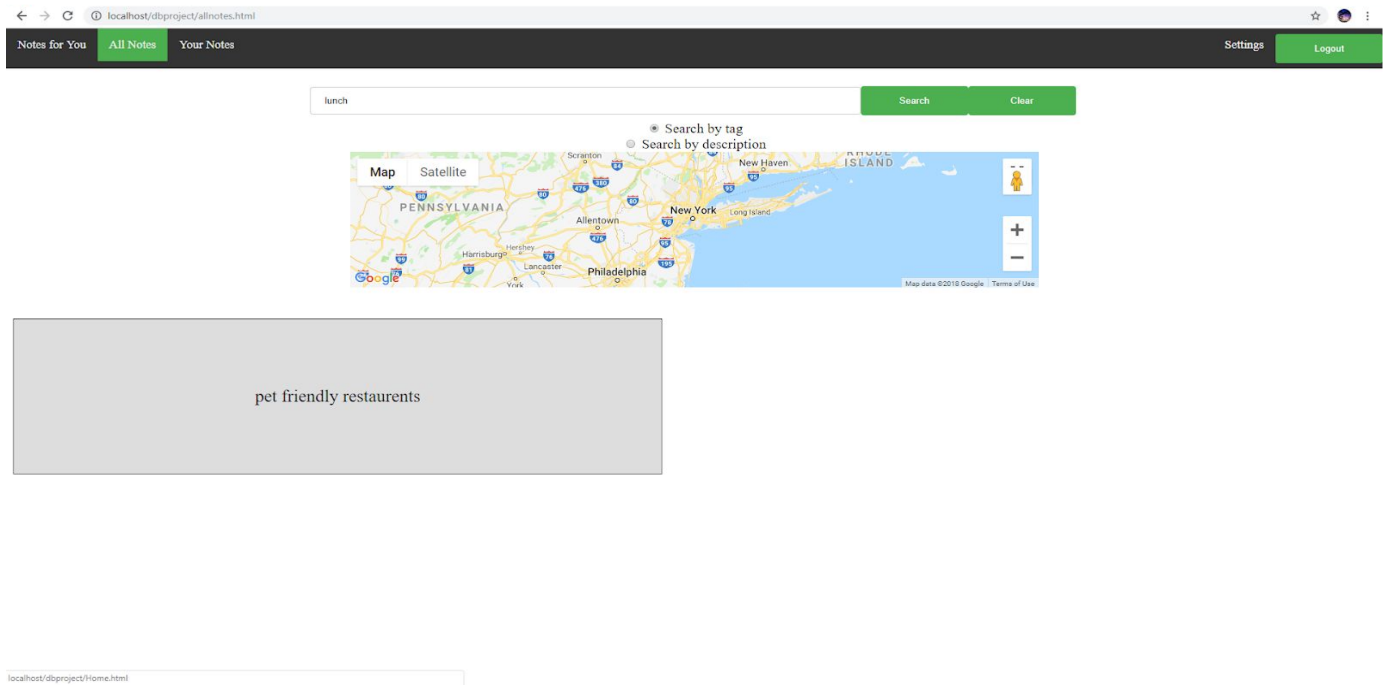


4. Allnotes Tab, user must search notes using tags or description and he has an option of selecting location around which notes within a radius of 100km will be displayed.



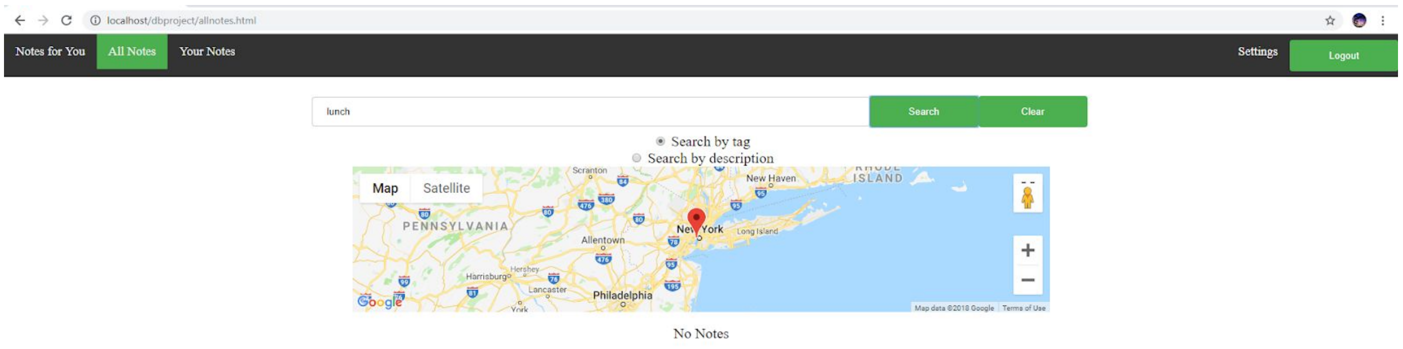
Search above to see notes

5. Notes with food as tag being displayed, No location input given.

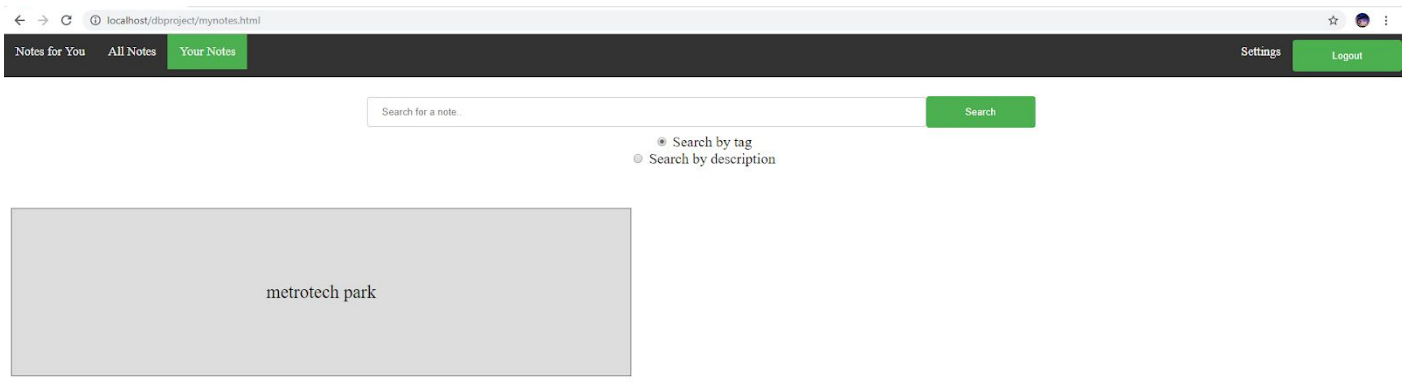


pet friendly restaurents

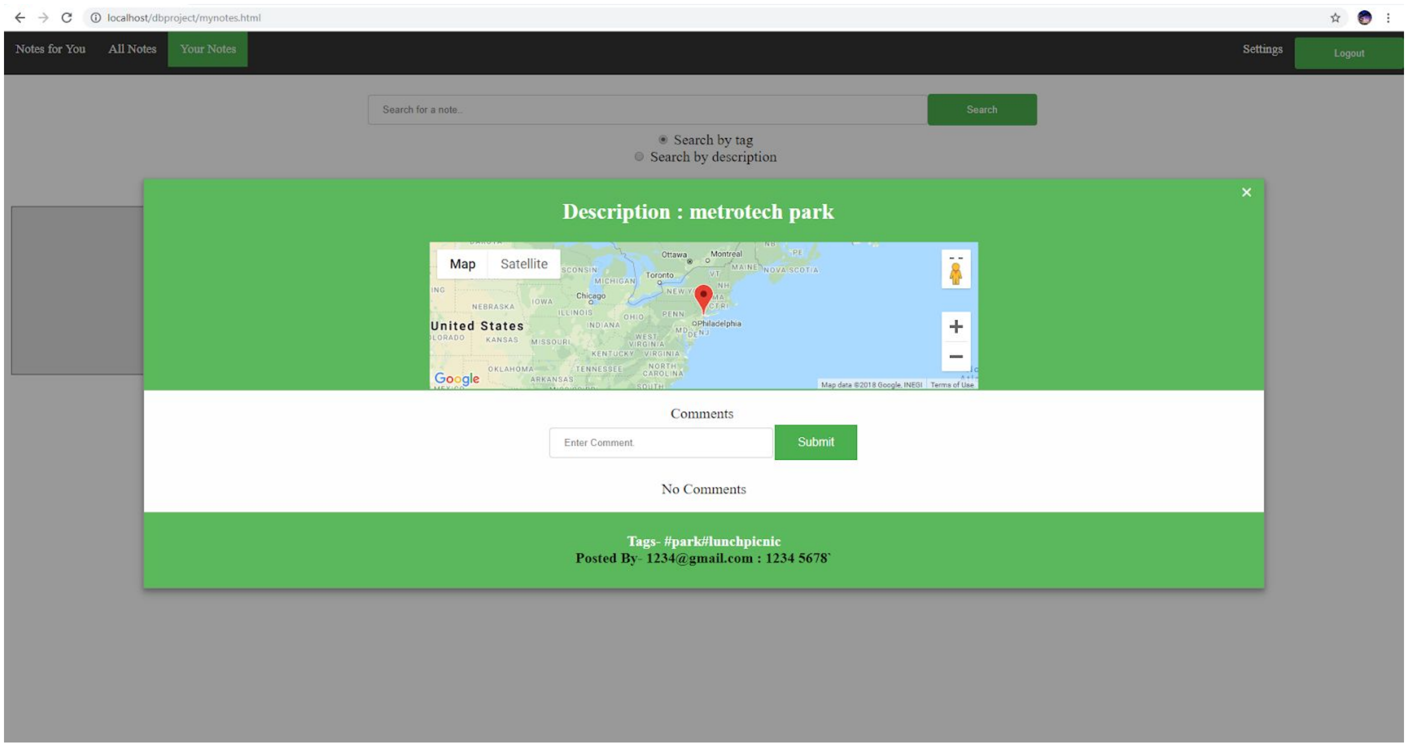
6. Notes around a location, Here no notes around the selected location.



7. Your Notes Tab, displaying notes created by current user



8. When user clicks on a note, a window pops up showing more details on the note selected.


localhost/dbproject/mynotes.html

Notes for You | All Notes | Your Notes | Settings | Logout

Search for a note.. | Search

● Search by tag
○ Search by description

**Description : metrotech park**

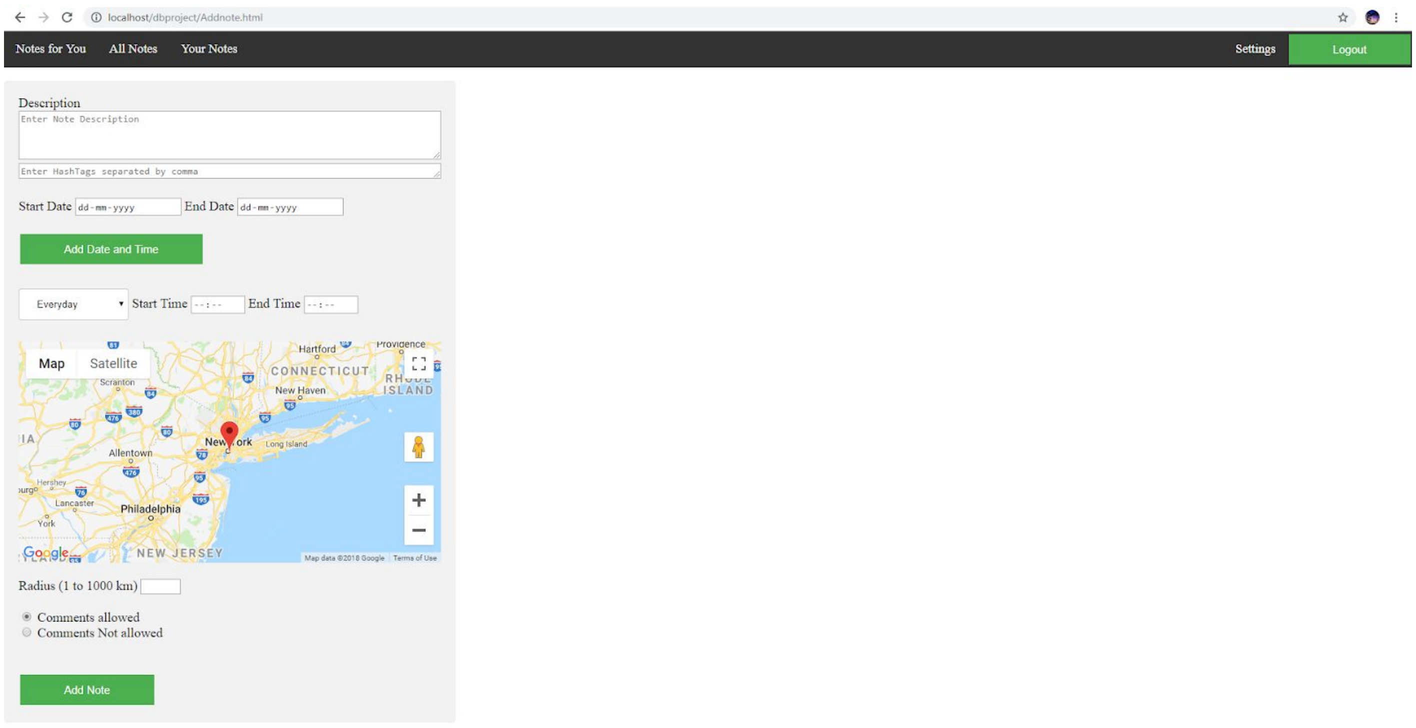
Comments

Enter Comment | Submit

No Comments

**Tags- #park#lunchpicnic**
**Posted By- 1234@gmail.com : 1234 5678`**

9. Add Note Page, available in the settings dropdown box.


localhost/dbproject/Addnote.html

Notes for You | All Notes | Your Notes | Settings | Logout

Description
Enter Note Description
Enter HashTags separated by comma

Start Date dd-mm-yyyy End Date dd-mm-yyyy

Add Date and Time

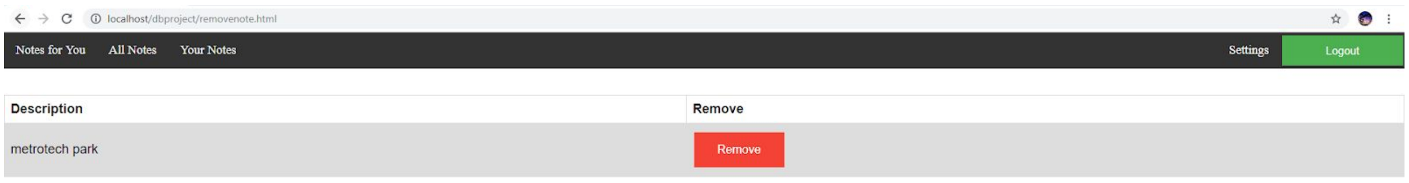
Everyday Start Time --:-- End Time --:--

Radius (1 to 1000 km)

● Comments allowed
○ Comments Not allowed

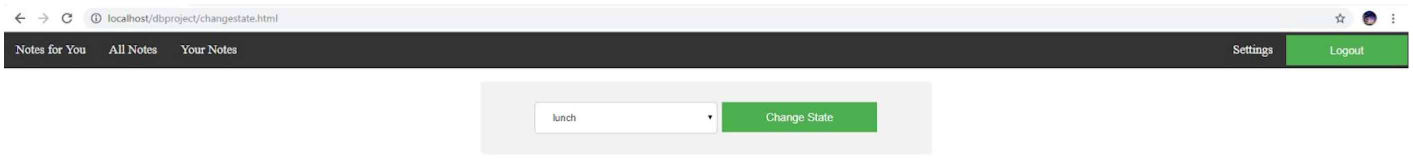
Add Note

10. Remove Note Page, available in the settings dropdown box

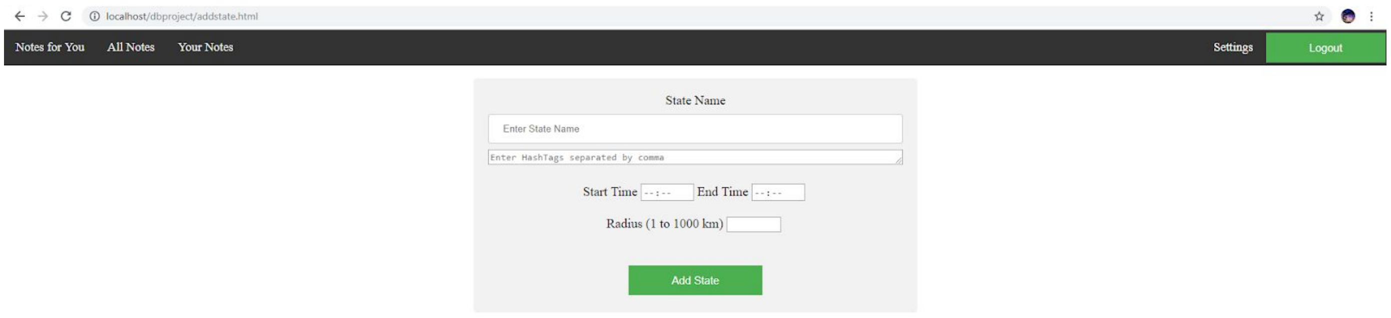


| Description | Remove |
|---|---|
| metrotech park | Remove |

11. Change State Page, available in the settings dropdown box



12. Add State Page, available in the settings dropdown box


localhost/dbproject/addstate.html

State Name

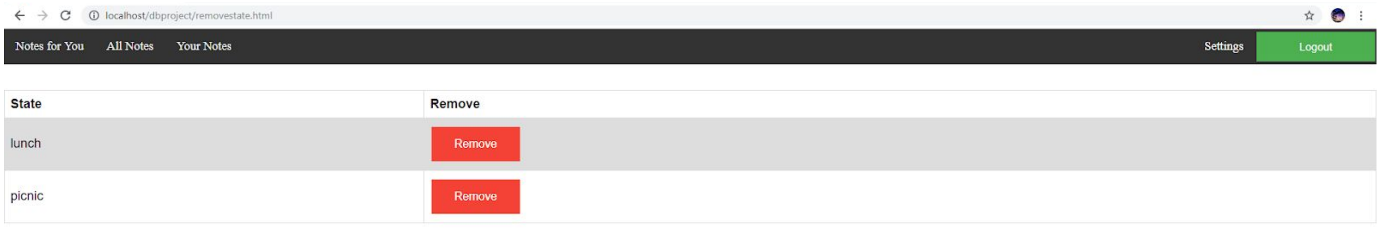
Enter State Name

Enter HashTags separated by comma

Start Time --:-- End Time --:--

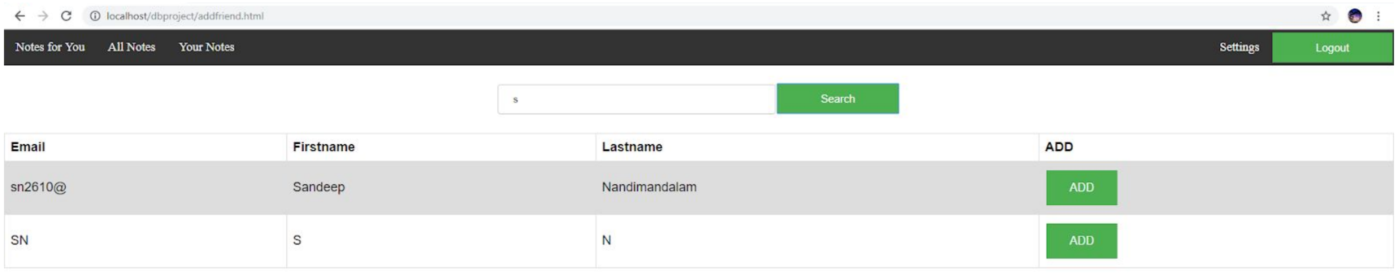
Radius (1 to 1000 km)

13. Remove State Page, available in the settings dropdown box


localhost/dbproject/removestate.html

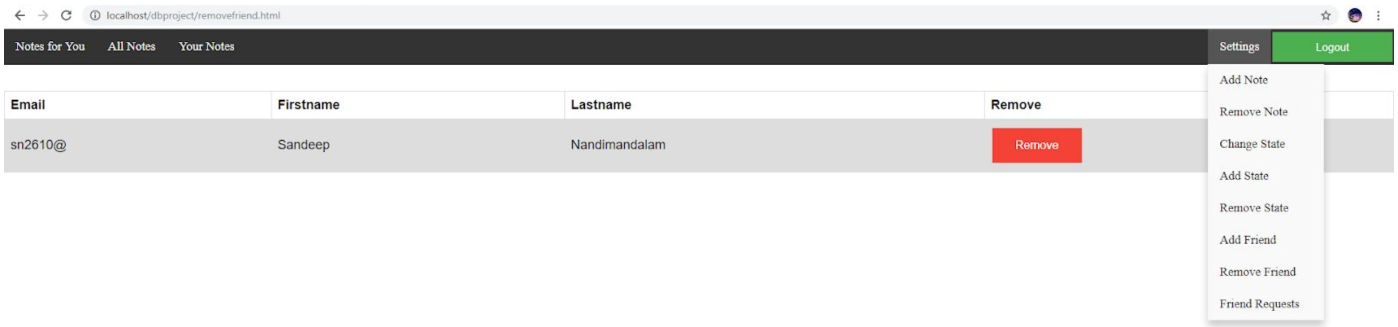
| State | Remove |
|---|---|
| lunch | Remove |
| picnic | Remove |

14. Add Friend Page, available in the settings dropdown box. The user can search for people by entering a name in the search bar.



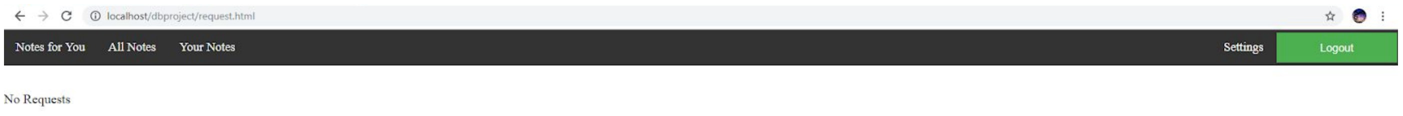
| Email | Firstname | Lastname | ADD |
|---|---|---|---|
| sn2610@ | Sandeep | Nandimandalam | ADD |
| SN | S | N | ADD |

15. Remove Friend Page, available in the settings dropdown box (Settings Dropdown shown)



| Email | Firstname | Lastname | Remove |
|---|---|---|---|
| sn2610@ | Sandeep | Nandimandalam | Remove |

16. Friends Requests Page, available in the settings dropdown box



No Requests