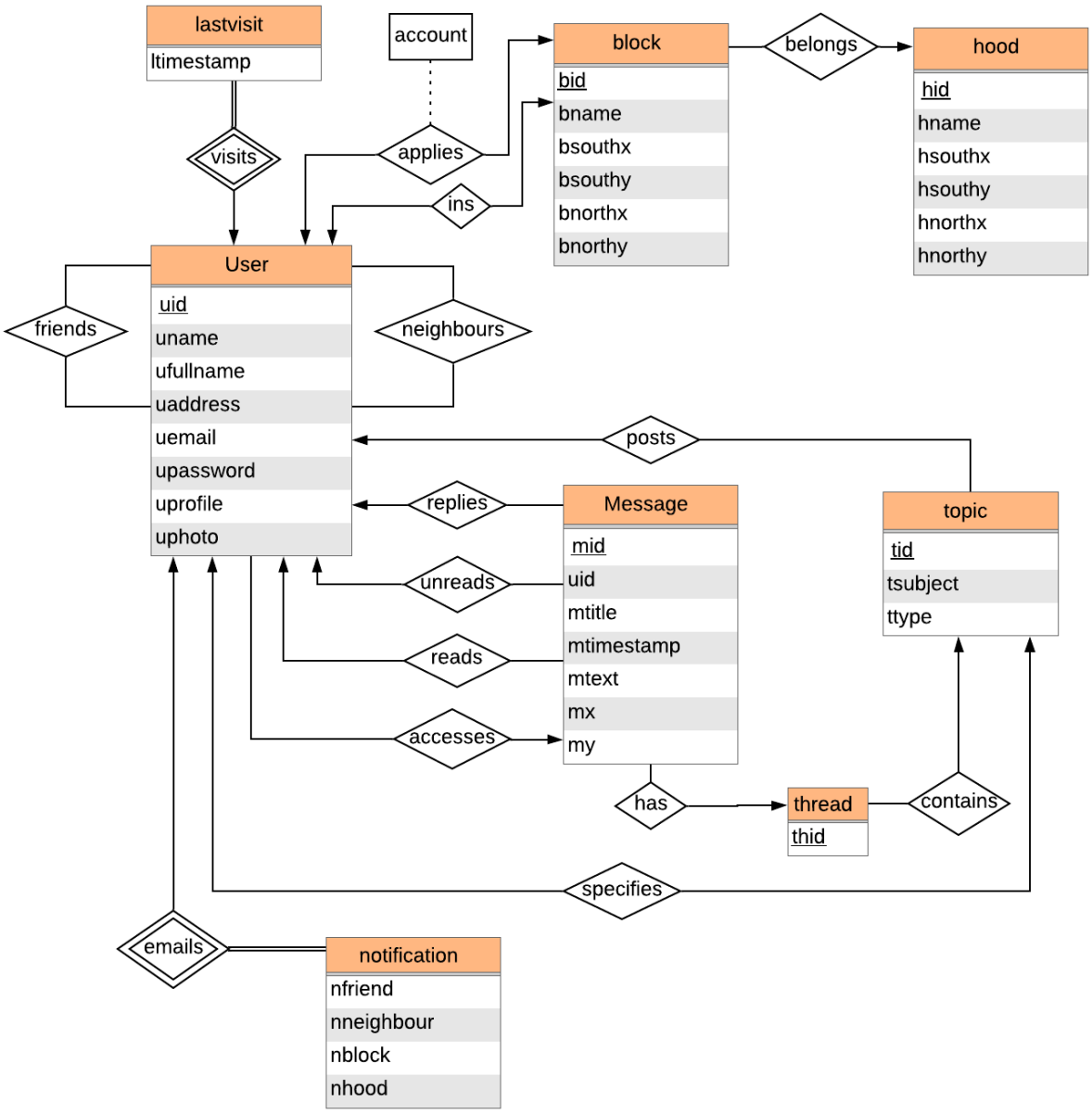
Introduction

This project is based for a website that allows people to communicate with other people that live in their neighborhood. The users can sign up for the service and specify where they live; they can then send and receive messages to other users living close by, and participate in discussions with those users. We present the following ER diagram to represent our database.



We include the following Entities in our Entity – Relational diagram that models the project.

1. User – It stores the details about all the user. Each user is identified using a unique username, user's full name, user's address, user's email address, user's account password, a short profile where user can describe himself and a profile photo.

2. Block – It stores the details about all the blocks that are identified using a block id and axis-aligned rectangles that can be defined by location coordinates of Southwest and Northeast corner of the rectangle. We also store the name associated with that block.
3. Hood – It stores the details about all the hoods. The data organisation in this relation is similar to the data organisation of the block relation.
4. Message – It stores the details about all the messages that exchanged amongst the users. Each message is identified using a unique message id. We store the sender of the message, the message title, the message text, the timestamp when the message is sent and coordinates of location from where the message was sent.
5. Topic – It stores the details about various topics and each topic is identified using a unique topic id. We store the subject and type of the topic. The type specifies the accessibility of that topic indicating the type of people who can read and reply to that topic.
6. Thread – It stores all the threads which are present in the system and each thread is uniquely identified by a thread id.
7. LastVisit – lastvisit stores the timestamps of all users which specifies the last time when a user accessed the system.
8. Notification – It stores the notification preference of all users. It indicates whether a user prefers a notification when a content is posted by a friend, neighbour, block member and/or a hood member.

We include the following relations in our Entity – Relational diagram that models the project.

1. Neighbours, one user can be neighbour of many other users.
2. Friends, one user can be friends with many other users.
3. Applies, each user submits an application for a particular block and acount maintains the number of people who have accepted the application.
4. Ins, specifies the block that a particular user belongs to.
5. Belongs, many blocks belong to a single hood.
6. Visits, each user has a timestamp which specifies the last time when that user used the system.
7. Posts, one user can create a number of topics.
8. Replies, one user can reply to many messages.
9. Unreads, each user has different unread messages.
10. Reads, each user has different read messages. Since, a message can be read and unread by many users depending on the message's accessibility, we maintain 2 different relations to store the read and unread messages.
11. Accesses, each message can be accessible to many users and is used to specify the users to whom a particular message is accessible.
12. Has, specifies the thread that a particular message belongs to.
13. Contains, each topic contains many threads.
14. Specifies, each user defines the type of the topic. The user specifies if the topic is visible to his friends, neighbours, block members and/or hood members.
15. Emails, each user has its own preference for receiving an email notification when a new content is posted.

Following is the equivalent relational schema of the above ER diagram.

1. user (uid, uname, ufullname, uaddress, uemail, upassword, uprofile, uphoto)
   uid,uname is the primary key.
2. hood (hid, hname, hsouthx, hsouthy, hnorthx, hnorthy)
   hid is the primary key.
3. block (bid, bname, bsouthx, bsouthy, bnorthx, bnorthy)
   bid is the primary key.
4. message (mid, uid, mtitle, mtimestamp, mtext, mx, my)
   mid is the primary key.
   uid references uid in user.
5. topic (tid, tsubject, ttype)
   tid is the primary key.
6. thread (thid)
   thid is the primary key.
7. notification (uid, nfriend, nneighbor, nblock, nhood)
   uid reference uid in user.
8. lastvisit (uid, ltimestamp)
   uid reference uid in user.
9. applies (uid, bid, acount)
   uid, bid is the primary key.
   uid reference uid in user.
   bid references bid in block.
10. ins (uid, bid)
    uid, bid is the primary key.
    uid reference uid in user.
    bid references bid in block.
11. belongs (hid, bid)
    hid, bid is the primary key.
    hid references hid in hood.
    bid references bid in block.
12. friends (uid, fid)
    uid, fid is the primary key.
    uid reference uid in user.
    fid references uid in user.
13. neighbours (uid, nid)
    uid, nid is the primary key.
    uid reference uid in user.
    fid references uid in user.
14. posts (uid, tid)
    uid, tid is the primary key.
    uid references uid in user.
    tid references tid in topic.

15. contains (tid, thid)
    tid, thid is the primary key.
    tid references tid in topic.
    thid references thid in thread.
16. has (thid, mid)
    thid, mid is the primary key.
    thid references thid in thread.
    mid references mid in message.
17. replies (uid, mid)
    uid, mid is the primary key.
    uid references uid in user.
    mid references mid in message.
18. unreads (uid, mid)
    uid, mid is the primary key.
    uid references uid in user.
    mid references mid in message.
19. reads (uid, mid)
    uid, mid is the primary key.
    uid references uid in user.
    mid references mid in message.
20. accesses (uid, mid)
    uid, mid is the primary key.
    uid references uid in user.
    mid references mid in message.
21. specifies (tid,uid)
    tid, uid is the primary key.
    tid references topic id in topic.
    uid references uid in user.

Sample Data

mid	uid	mtitle	mtimestamp	mtext	mx	my
1	2	How to feed a dog	2019-11-25 00:00:00	Do you have a dog?	0	0
2	3	How to feed a dog	2019-11-25 09:00:00	No. But I like dog!	0	0
3	2	About your name	2019-11-25 00:01:00	Your name is so cooool!	0	0
4	5	About your name	2019-11-25 08:01:00	Thank you so much :)	0	0
5	4	About your home address	2019-11-25 00:01:00	Where do you live? I cannot find your address ...	0	0
6	2	Where can I wash clothes	2019-11-25 03:00:00	Someone knows?	0	0
7	2	Rent a car	2019-11-25 03:00:00	I want to rent a car and I will pay $1000 for a d...	0	0
8	2	About car accident	2019-11-25 06:00:00	Where is the bicycle accident happenned?	0	0
9	2	About car accident	2019-11-25 06:00:00	Really? Sounds terrible!	0	0
10	2	Basketball match	2019-11-25 09:00:00	Does anyone like playing basketball?	0	0

Message

uid	bid	acount
1	1	0

Applies

uid	nid
2	4

Neighbours

mid	uid
10	3
5	4
6	4
8	4
9	4
10	4
3	5
4	5
8	5
9	5
10	5
8	6
9	6
10	6
8	7
9	7

Accesses

tid	uid
1	2
1	3
1	5
2	2
2	4
3	2
3	3
3	4
4	2
4	3
4	4

Specifies

thid	mid
1	1
1	2
2	3
2	4
3	5
4	6
5	7
6	8
6	9
7	10

Has

uid	bid
2	1
3	1
4	1
5	2
6	2
7	2
8	3
9	3

Ins

hid	bid
1	1
1	2
2	3

Belongs

uid	tid
2	1
2	2
2	3
2	4

Posts

tid	thid
1	1
1	2
2	3
3	4
3	5
4	6
5	7

Contains

uid	ltimestamp
1	2019-11-25 00:00:00
2	2019-11-25 02:00:00
3	2019-11-25 00:00:00
4	2019-11-25 00:00:00
5	2019-11-25 00:00:00
6	2019-11-25 00:00:00
7	2019-11-25 00:00:00
8	2019-11-25 00:00:00
9	2019-11-25 00:00:00

LastVisit

uid	fid
3	2
5	2
2	3
2	5

Friends

bid	bname	bsouthx	bsouthy	bnorthx	bnorthy
1	68th-74th street	0	0	0	0
2	80th-91th street	0	0	0	0
3	3rd-9th street	0	0	0	0

Block

thid
1
2
3
4
5
6
7

Threads

hid	hname	hsouthx	hsouthy	hnorthx	hnorthy
1	Bay Ridge	0	0	0	0
2	Park Slope	0	0	0	0

Hood

mid	uid	mtitle	mtimestamp	mtext	mx	my
1	2	How to feed a dog	2019-11-25 00:00:00	Do you have a dog?	0	0
2	3	How to feed a dog	2019-11-25 09:00:00	No. But I like dog!	0	0
3	2	About your name	2019-11-25 00:01:00	Your name is so cooool!	0	0
4	5	About your name	2019-11-25 08:01:00	Thank you so much :)	0	0
5	4	About your home address	2019-11-25 00:01:00	Where do you live? I cannot find your address ...	0	0
6	2	Where can I wash clothes	2019-11-25 03:00:00	Someone knows?	0	0
7	2	Rent a car	2019-11-25 03:00:00	I want to rent a car and I will pay $1000 for a d...	0	0
8	2	About car accident	2019-11-25 06:00:00	Where is the bicycle accident happenned?	0	0
9	2	About car accident	2019-11-25 06:00:00	Really? Sounds terrible!	0	0
10	2	Basketball match	2019-11-25 09:00:00	Does anyone like playing basketball?	0	0

Message

uid	uname	ufullname	uaddress	uemail	upassword	uprofile	uphoto
1	Ares	Zuoyiwei Zhang		aczzyw@gmail.com	pwd1	A handsome boy	None
2	Chuck	Rochak Agrawal		24rochak@gmail.com	pwd2	A handsome boy too	None
3	Ricky	Boqi Tan		ricky@gmail.com	pwd3		None
4	Bob	Bobby Aloupis		bob@gmail.com	pwd4		None
5	Amy	Amy Suel		amy@gmail.com	pwd5		None
6	Tony	Tony Stark		tony123@gmail.com	pwd6		None
7	Babala	Crise Potter		cp1997@gmail.com	pwd7		None
8	Struke	Gary Lee		gary666@gmail.com	pwd8		None
9	Doglike	Anna Mellon		annapretty@gmail.com	pwd9		None

User

Sample Queries

Joining

Sign-up:

```
insert into `user` (`uid`, `uname`, `ufullname`, `uaddress`,
`uemail`, `upassword`, `uprofile`, `uphoto`)
values (1, 'Ares', 'Zuoyiwei Zhang', '', 'aczzyw@gmail.com',
'pwd1', 'A handsome boy', 'None');
```

Check if email has signed up or not:

```
select * from user where uemail = 'aczzyw@gmail.com';
```

Apply to be a member:

```
insert into `applies` (`uid`, `bid`, `acount`)
values (1, 1, 0);
```

Accept new block members:

```
update `applies`
set acount=(select `acount` from `applies` where `uid` =1) + 1
where uid = 1;
```

Update a profile:

```
update `user` set uprofile = 'add something', uphoto = 'url'
where uid = 1;
```

Content Posting

Starting a new topic:

```
insert into `topic` (`tid`, `tsubject`, `ttype`)
values (1, 'dog', 'friend'), (2, 'cat', 'neighbor'), (3, 'car',
'block'), (4, 'food', 'hood');

insert into `posts` (`uid`, `tid`) values (2, 1), (2, 2), (2,
3), (2, 4);
```

Specify who can chat under this topic:

```
insert into `specifies` (`tid`, `uid`)
values (2, 2), (3, 2), (4, 2);
```

Starting a new thread with an initial message and specifying who can access it:

```
insert into `thread` (`thid`) values (1);

insert into `contains` (`tid`, `thid`) values (1, 1);

insert into `message` (`mid`, `uid`, `mtitle`, `mtimestamp`,
`mtext`, `mx`, `my`) values (1, 1, 'How to feed a dog', '2019-
11-25 00:00:00', 'Do you have a dog?', 0, 0);

insert into `has` (`thid`, `mid`) values (1, 1);
insert into `accesses` (`mid`, `uid`) values (1, 1);
```

Replying to a message:

```
insert into `message` (`mid`, `uid`, `mtitle`, `mtimestamp`,
`mtext`, `mx`, `my`)
values (2, 2, 'A reply', '2019-11-25 01:00:00', 'Dogs like
meat.', 0, 0);

insert into `has` (`thid`, `mid`) values (1, 2);
insert into `accesses` (`mid`, `uid`) values (2, 1);
```

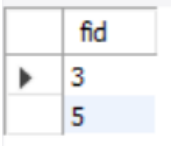
Friendship

add someone as a friend or neighbour:

```
insert into `friends` (`uid`, `fid`) values (1, 2), (2, 1);
insert into `neighbors` (`uid`, `nid`) values (1, 2);
```

List current friends:

```
select f1.fid
from friends as f1, friends as f2
where f1.uid = f2.fid and f1.fid = f2.uid and f1.uid = 1;
```



	fid
▶	3
	5

List current neighbors:

```
select nid from neighbors where uid = 1;
```

Browse and search messages

list all threads in a user's block feed that have new messages since the last time the user accessed the system last time the user accessed the system:

```
create view block_topic as
select tid
from topic natural join specifies
where ttype = 'block' and uid = 2;

create view block_thread as
select thid
from `contains` join block_topic
on `contains`.tid = block_topic.tid;

create view accessable_message as
select has.thid as thid, has.mid as mid
from (block_thread join has on block_thread.thid = has.thid)
join accesses on has.mid = accesses.mid
where uid = 2;

select thid
from accessable_message natural join message, lastvisit
where lastvisit.uid = 2 and ltimestamp < mtimestamp;
```

```
drop view block_topic;
drop view block_thread;
drop view accessable_message;
```

	thid
▶	4
	5

All threads in friend feed that have unread message:

```
create view friend_topic as
select tid
from topic natural join specifies
where ttype = 'friend' and uid = 2;

create view friend_thread as
select thid
from `contains` join friend_topic
on `contains`.tid = friend_topic.tid;

create view accessable_message as
select has.thid as thid, has.mid as mid
from (friend_thread join has on friend_thread.thid = has.thid)
join accesses on has.mid = accesses.mid
where uid = 2;

select lastvisit.uid, message.mid
from accessable_message natural join message, lastvisit
where lastvisit.uid = 2 and ltimestamp < mtimestamp;

drop view friend_topic;
drop view friend_thread;
drop view accessable_message;
```

	uid	mid
▶	2	2
	2	4

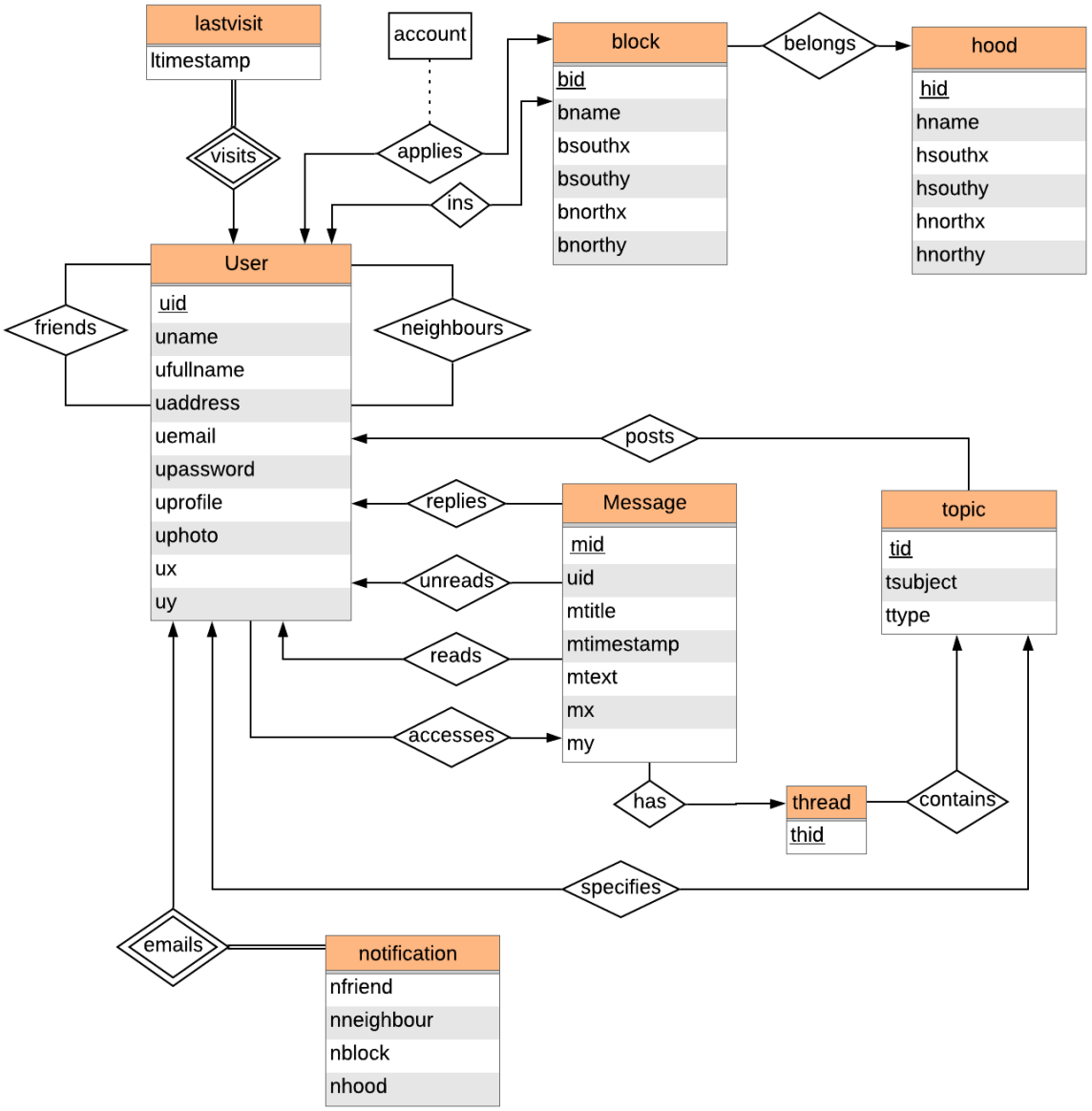
All messages containing the words “bicycle accident” across all feeds that the user can access:

```
select message.mid
from accesses join message on accesses.mid = message.mid
where accesses.uid = 2 and mtext like '%bicycle accident%';
```

	mid
▶	8

Updated Database Schema

During the implementation phase, few extra features were added and the updated database schema has been presented below.



The changes include adding the Geo-coordinates of the user's home. This would allow a particular user to view the home location of his friends and neighbours.

System Technical Details

- The system has been implemented as a web application. Currently, it is deployed on a locally hosted Node.js server.
- The programming languages that were used in the development include JavaScript, HTML and CSS.
- For, the web application, Express.js framework was used to develop single page applications and make calls to the server.
- The database system that was used for storing data is MySQL 8.
- Google maps API is used for getting the map data.

System Components

Login Page

The logic page is the entry point to enter the system. The user has to enter his username and password to enter the system.
If the entered data matches the data in the system, the user is granted access otherwise he is denied the access.
If the user is a new user, he can register himself to use the system by clicking the register button.

User Profile

The system allows users to create a profile of their own. If a user is not currently registered in the system, he/she can register by providing their email address, a username and a password. Apart from this, the user has the option to enter their Full name and a small description about themselves, where they can tell other users about themselves.
To get easily identified by other members of their block or hood, each user can also upload a profile picture.
The user has the option of storing their home address as well so that other users living in the nearby area can connect with them as their neighbours.
In addition to that a user can become friends with other users. The user can send a friend request to other user and if the other user accepts their friend request, they become friends of each other.
The user can submit the request to apply to a new block from his user profile. This is done by selecting a block from the drop-down menu. When a user posts a new application, all the users living in that block are notified about the request in their user profile page where they can accept or deny the request.

Chat Page

The chat page allows the user to interact with other users. These users include friends, neighbours, block members and hood members.
When the user logs in, he can see the unread messages highlighted in green colour.
The user can read, reply to all messages, threads, topics that he has access to. The access rights are defined by the owner of the topic, thread and message individually.
The user can also initiate a new thread or a topic.
If a user initiates a topic, he has to specify the accessibility of that topic. The topic can be accessible by all his friends, neighbours, block members or hood members.
If the user initiates a new thread, he has to specify that topic to which the thread belongs to. He can then specify the accessibility of that thread. The accessibility of the thread is a subset of the accessibility of the topic. The user can specify finer details about the accessibility of the thread. He can choose between an individual friend, all his friends, his neighbours or all block members. In addition, has to specify the title of the thread.

The user can also search for messages by entering a keyword in the search bar. All the accessible messages that contain the keyword will be displayed to the user after the search button is clicked.

Map view

The system incorporates a map view where each user can the location of the recent messages of all threads that he has access to.
The map view also allows the user to filter the messages based on their locations. To use this functionality, the user clicks on the *search* button. The user is then presented a rectangle which he can use to select a region based on his preference. The user can resize the rectangle, move the rectangle to any location of his preference. After the desired region has been selected, the user clicks on the *locate* button to view the messages that he has access to that were sent from the selected region.
Clicking on the legend toggles the visibility of the highlighted hood area and block areas that belong to the user's hood. The user can read the recent messages directly from the map view page. If a marker is clicked then the most recent unread message from that user is displayed.

Contacts

Contacts allows a user to see their contacts. This includes his friends, neighbours, block members and hood members.
A user can see the home location of their friends and neighbours. The locations marked on the map are colour coded for quick identification. The friends are marked using a red marker whereas the neighbours are marked using a yellow marker.
The user can use the legend to toggle the visibility of the markers of friend and neighbours. Similar to the map view, clicking on the legend also toggles the visibility of the highlighted hood area and block areas that lie within the user's hood.
The user can click on a name of a contact in the contact list to view the profiles of that contact. The profile details that are displayed include profile picture, username, email address, home address of the user.
If the user wishes to add the contact as a friend, they can send a friend request through the profile. If a friend request has already been sent, the user is notified accordingly. This avoids sending repetitive requests and avoids storing redundant data in the database.

Implementation Details

- When a user sends a friend request to another user, we add a tuple (uid, fid) to the friends' relation. This indicates that user having the usersid as uid wants to be friends with another user having userid as *fid.* When the user with userid *fid* accepts the request, the system add a tuple (fid, uid) to the friends' relation. Now, since, both the tuples (uid, fid) and (fid, uid) are present in the friends' relation, we declare the users as friends. If the user with userid *fid* rejects the request, the system does not add the tuple (fid, uid) to the friends' relation and also removes the tuple (uid, fid) from the friends' relation indicating that they cannot become friends.

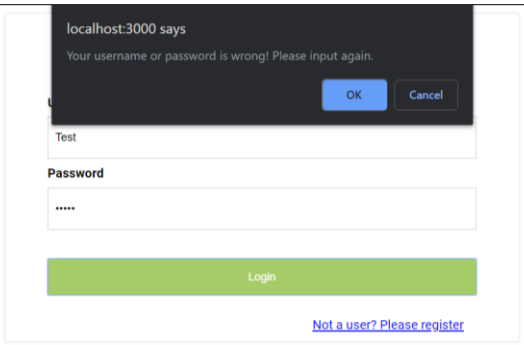
- Whenever, a user applies to a new block, a tuple (uid, bid, acount) is added to the relation applies. Uid indicates the userid of the user that is applying to the block with block id bid. *Acount* indicates the number of people who have accepted the request. Whenever a user that lives in block having block id as bid; accepts the request, the system increments the value of *account* by 1. If the number of people living in the block is greater than 3, then we declare the request as a successful request if value of *acount* becomes 3. If the number of people living in the block is less than or equal to 3, then we declare the request as a successful request if value of *acount* becomes equal to number of people living in the block.

- When a user logs out of the system, we add a tuple (uid, ltimestamp) to the lastVisit relation. It indicates the last time when user with userid uid logged out of the system. This data is used to find out which messages are unread messages. All the accessible messages by the user that have a timestamp greater than the ltimestamp of that user are classified as unread messages and the system highlights the threads containing those messages with green colour.

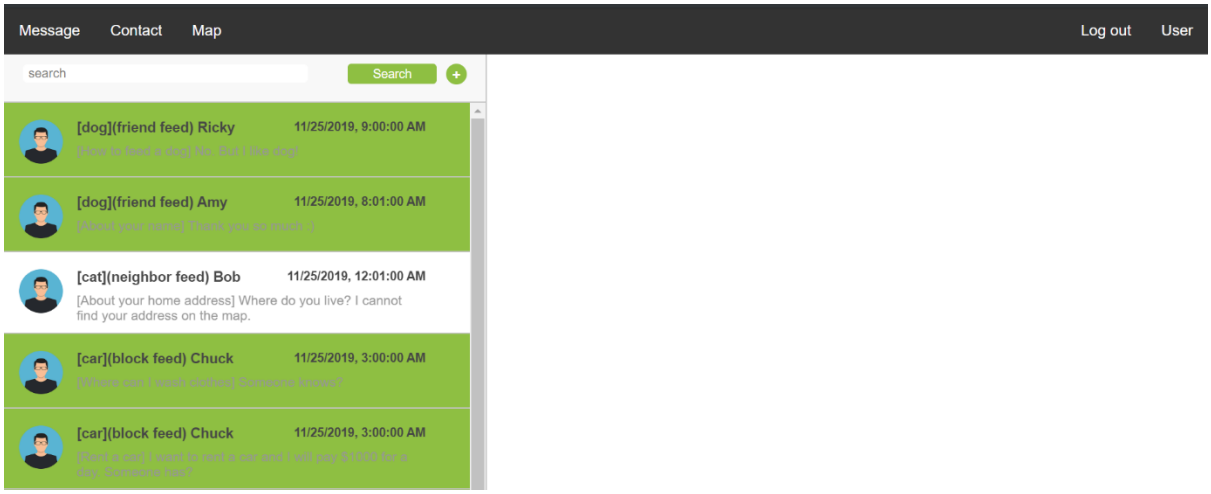
- The system makes calls to database using prepared statements. This help in avoiding SQL injection to the database. Also, the input to the database query is sanitized. The server code never sends data received from database directly to the client browser. These precautions make the system safe from cross site scripting (XSS and CSRF) attacks.

- The password is not stored directly in the database. The system utilizes one way hashing to store the password. Specifically, hashing with base 64 is used for storing passwords. This maintain the system secure, even if a hacker tries to gain access to the password using the Man in the middle attacks.

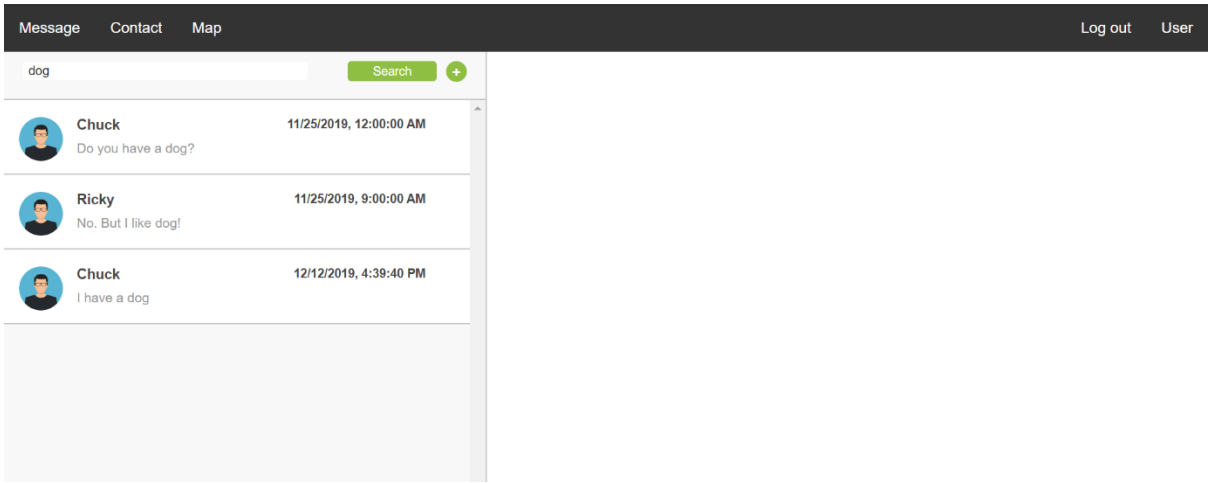
Sample Screenshots



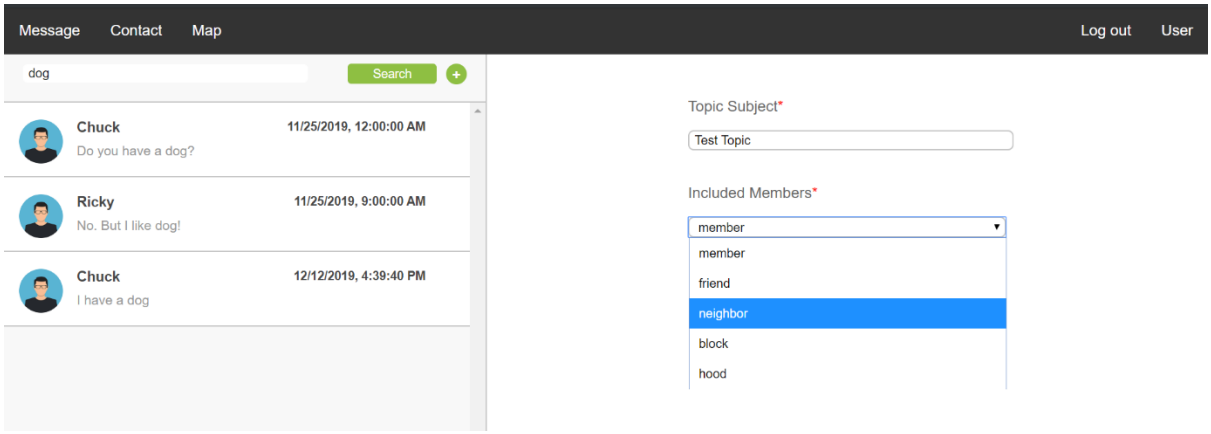
Failed attempt on the login Page



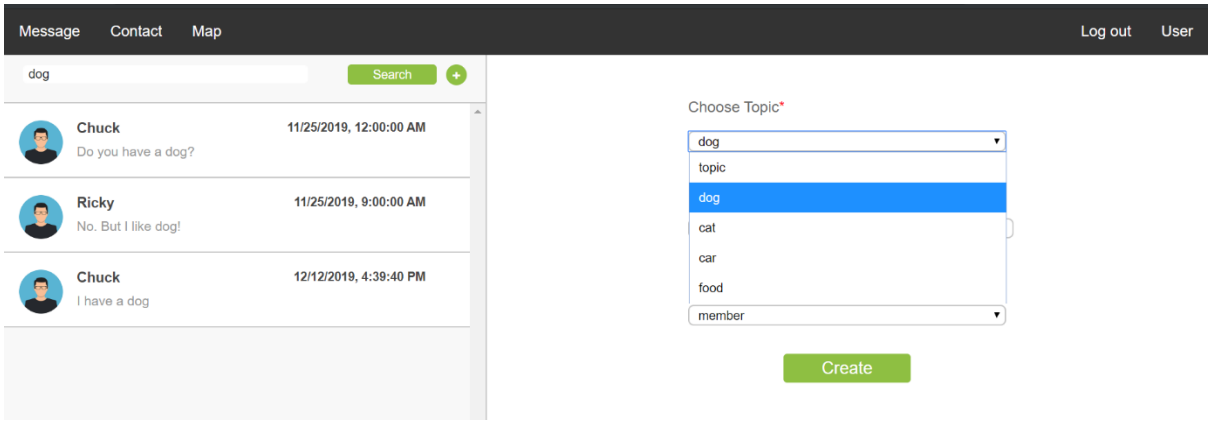
Unread messages highlighted in green colour



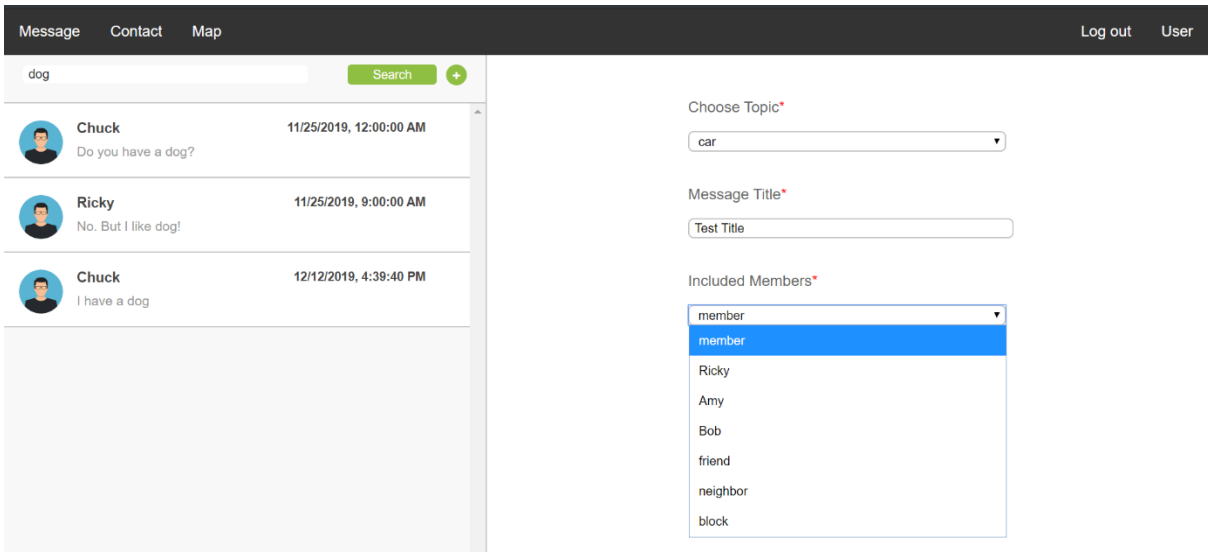
Working of the search tool



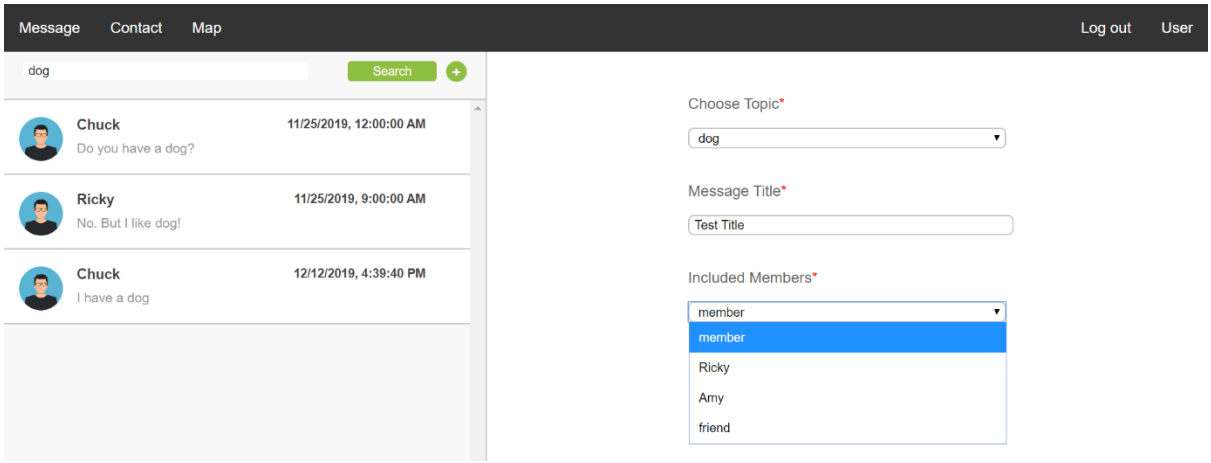
Creating new topic and defining its accessibility



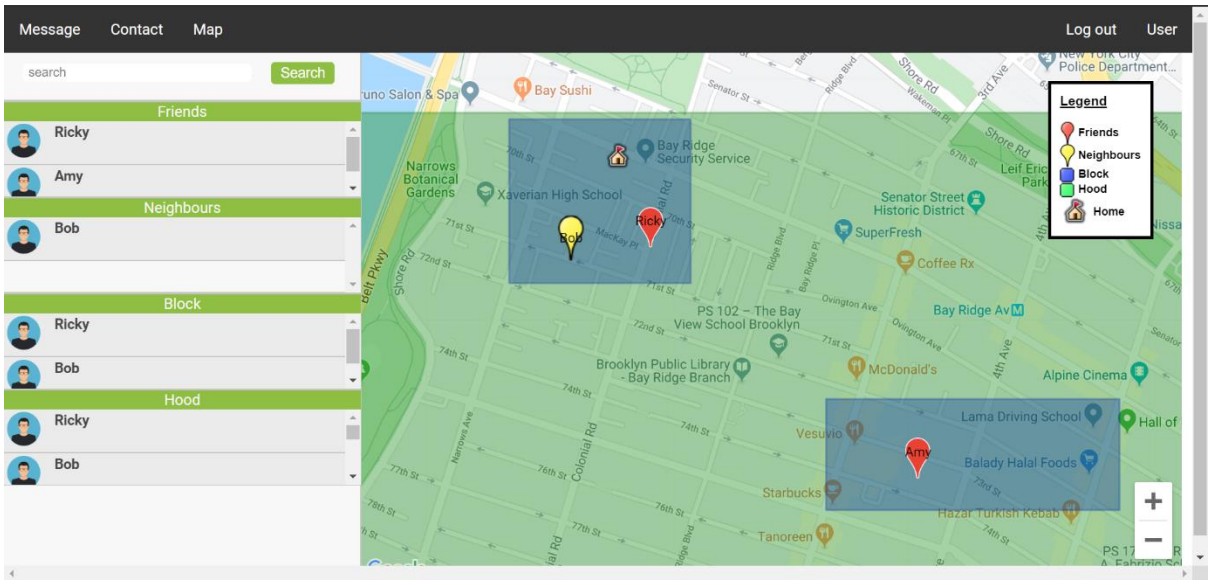
Choosing accessible topics while creating new thread



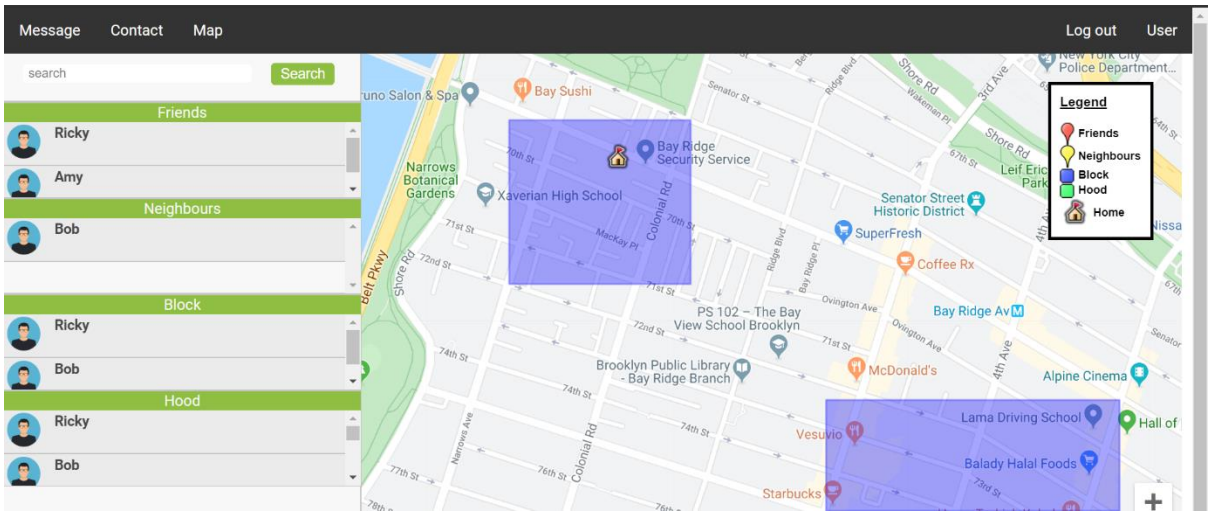
Defining accessibility of the new thread based on selected topic's accessibility



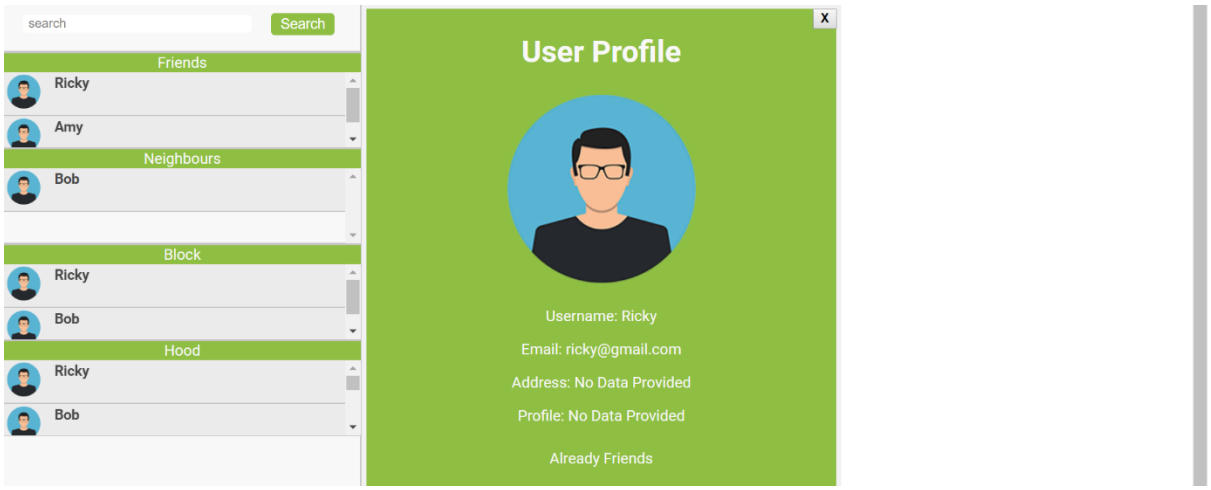
Defining accessibility of the new thread based on selected topic's accessibility



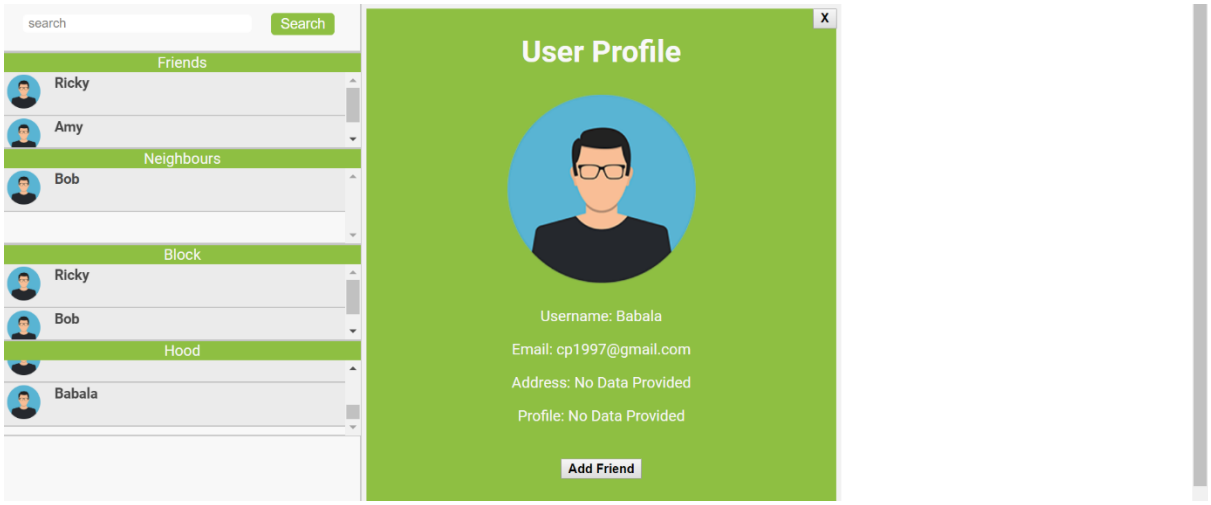
Contacts page



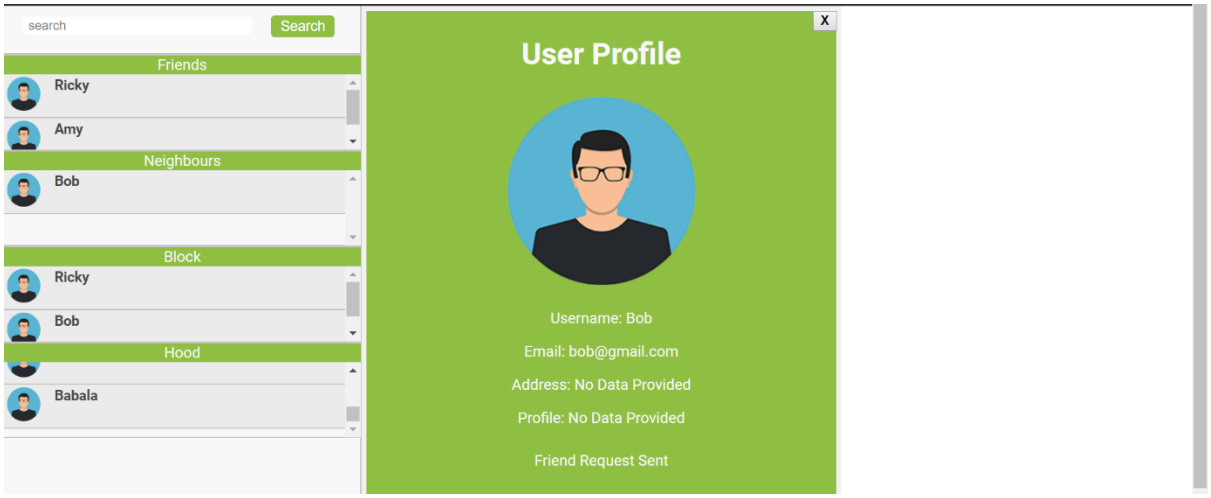
Toggling various markers using legend



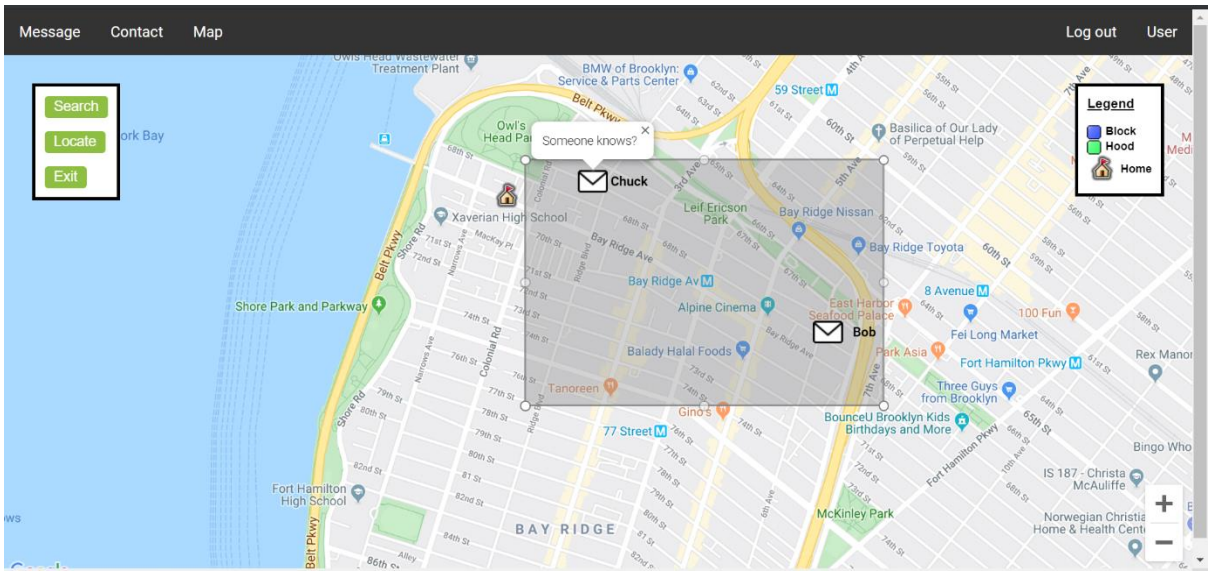
Viewing user profile of a friend



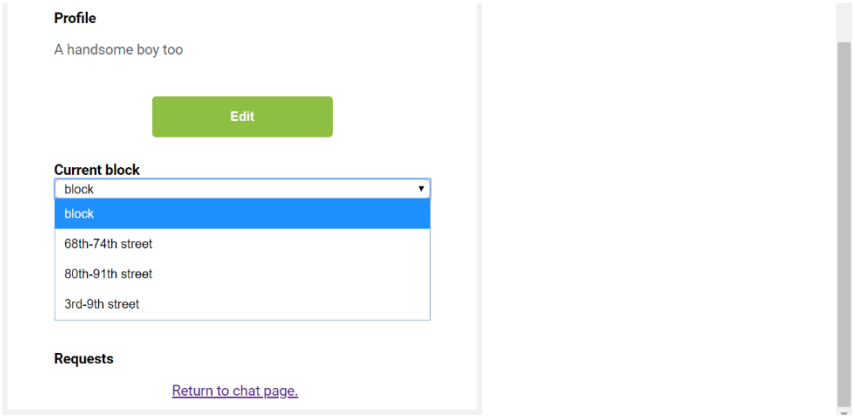
Viewing user profile of a user who is not friend



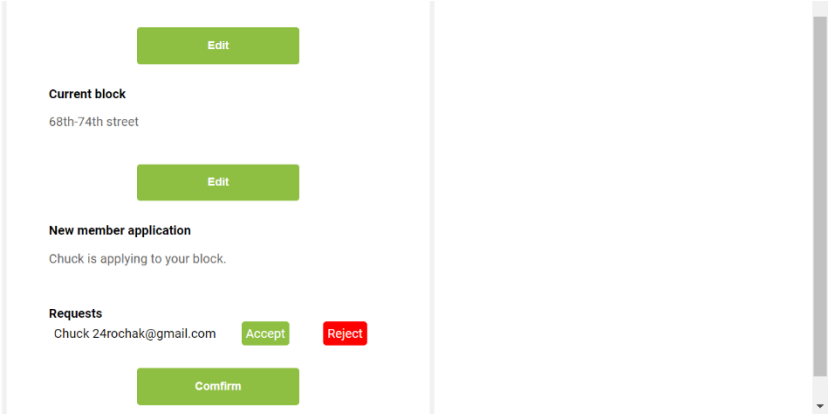
Viewing user profile of a user to whom friend request has been sent



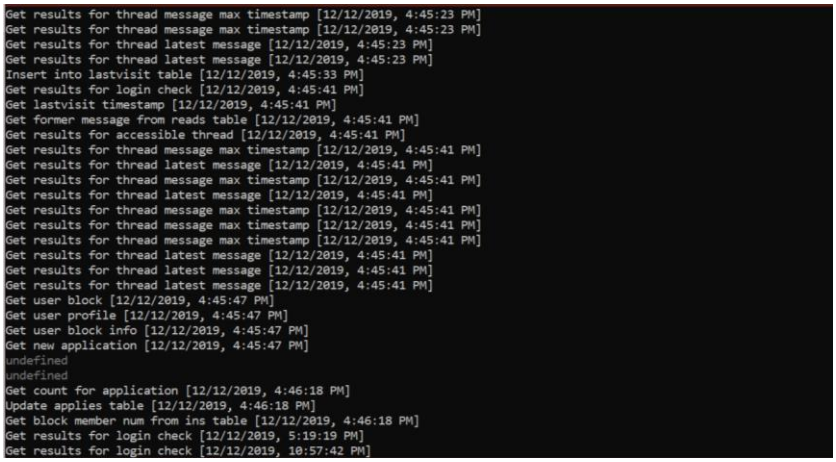
Working of searching messages using location and viewing recent messages in map view.

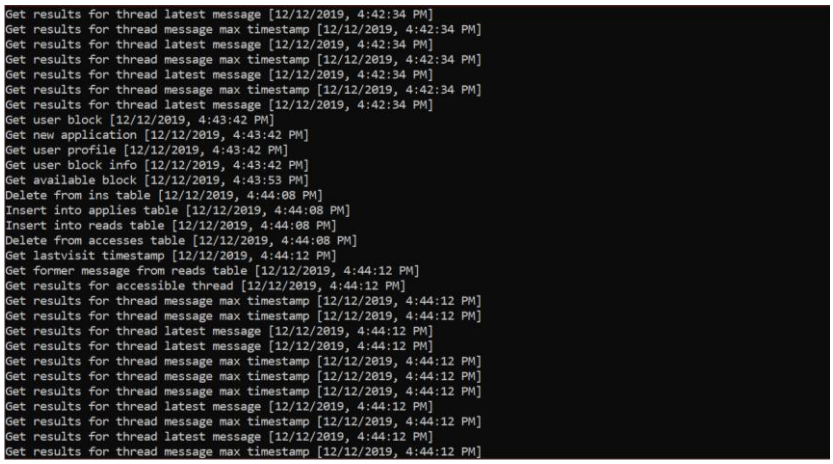


Applying to a new block in the user profile page



Incoming friend request and new block member request


Get results for thread message max timestamp [12/12/2019, 4:45:23 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:23 PM]
Get results for thread latest message [12/12/2019, 4:45:23 PM]
Get results for thread latest message [12/12/2019, 4:45:23 PM]
Insert into lastvisit table [12/12/2019, 4:45:33 PM]
Get results for login check [12/12/2019, 4:45:41 PM]
Get lastvisit timestamp [12/12/2019, 4:45:41 PM]
Get former message from reads table [12/12/2019, 4:45:41 PM]
Get results for accessible thread [12/12/2019, 4:45:41 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:41 PM]
Get results for thread latest message [12/12/2019, 4:45:41 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:41 PM]
Get results for thread latest message [12/12/2019, 4:45:41 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:41 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:41 PM]
Get results for thread message max timestamp [12/12/2019, 4:45:41 PM]
Get results for thread latest message [12/12/2019, 4:45:41 PM]
Get results for thread latest message [12/12/2019, 4:45:41 PM]
Get results for thread latest message [12/12/2019, 4:45:41 PM]
Get user block [12/12/2019, 4:45:47 PM]
Get user profile [12/12/2019, 4:45:47 PM]
Get user block info [12/12/2019, 4:45:47 PM]
Get new application [12/12/2019, 4:45:47 PM]
undefined
undefined
Get count for application [12/12/2019, 4:46:18 PM]
Update applies table [12/12/2019, 4:46:18 PM]
Get block member num from ins table [12/12/2019, 4:46:18 PM]
Get results for login check [12/12/2019, 5:19:19 PM]
Get results for login check [12/12/2019, 10:57:42 PM]


Get results for thread latest message [12/12/2019, 4:42:34 PM]
Get results for thread message max timestamp [12/12/2019, 4:42:34 PM]
Get results for thread latest message [12/12/2019, 4:42:34 PM]
Get results for thread message max timestamp [12/12/2019, 4:42:34 PM]
Get results for thread latest message [12/12/2019, 4:42:34 PM]
Get results for thread message max timestamp [12/12/2019, 4:42:34 PM]
Get results for thread latest message [12/12/2019, 4:42:34 PM]
Get user block [12/12/2019, 4:43:42 PM]
Get new application [12/12/2019, 4:43:42 PM]
Get user profile [12/12/2019, 4:43:42 PM]
Get user block info [12/12/2019, 4:43:42 PM]
Get available block [12/12/2019, 4:43:53 PM]
Delete from ins table [12/12/2019, 4:44:08 PM]
Insert into applies table [12/12/2019, 4:44:08 PM]
Insert into reads table [12/12/2019, 4:44:08 PM]
Delete from accesses table [12/12/2019, 4:44:08 PM]
Get lastvisit timestamp [12/12/2019, 4:44:12 PM]
Get former message from reads table [12/12/2019, 4:44:12 PM]
Get results for accessible thread [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread latest message [12/12/2019, 4:44:12 PM]
Get results for thread latest message [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread latest message [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]
Get results for thread latest message [12/12/2019, 4:44:12 PM]
Get results for thread message max timestamp [12/12/2019, 4:44:12 PM]

Server logs