Database Project

Neighbor Chat

NYU Tandon - CS6083 - Fall 2019

Wenzhou Li wl2154

Linyi Yan ly1333

Contents

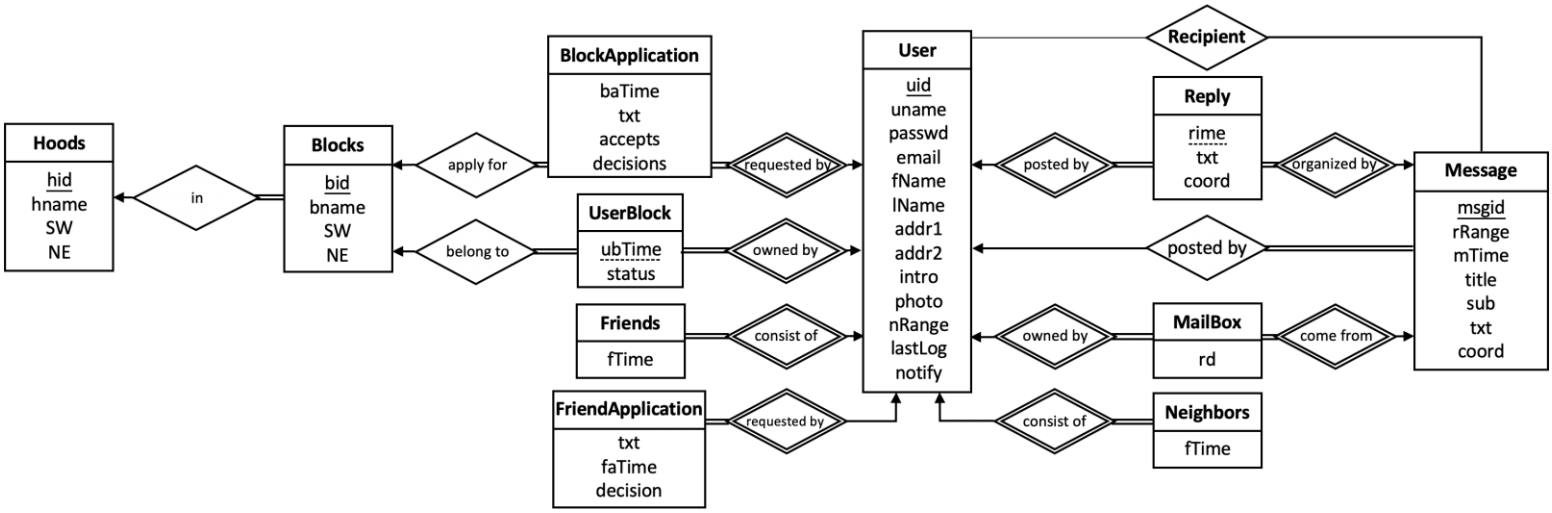
Part I - Relational Backend Design

a) Relational Schema & E-R Diagram

This part describes and justifies how we design the project in detail

i. E-R Diagram

This part is the E-R diagram for the project



ii. Relational Schema

This part shows the final relational schema we designed.

Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)

Hoods(hid, hname, SW, NE)

Blocks(bid, bname, hid, SW, NE)

UserBlock(uid, ubTime, bid, status)

BlockApplication(applicant, bid, baTime, txt, accepts, decisions)

Friends(uidA, uidB, fTime)

FriendApplication(applicant, recipient, txt, faTime, decision)

Neighbors(uidA, uidB, nTime)

Message(msgid, author, rRange, mtime, title, sub, txt, coord)

Recipient(msgid, uid)

MailBox(uid, msgid, rd)

Reply(msgid, uid, rTime, txt, coord)

iii. Schema Description

This part describes relational schema in detail.

Users: Personal information
Primary key: uid
uid: unique id identifies each user
uname, passwd, email, fName, lName: name, password, email, first name, last name of user
addr1, addr2: addr1: street and number, P.O. box, c/o.; addr2: apartment, suite, unit, building, floor, etc.
intro, photo: introduction of user and photo path in server default null when sign up, can be added later
nRange: neighbor range settings, limited to: same building(0) / block(1) / hood(2), default 2
lastLog: last log-out timestamp, update it each time user logs out, default current_timestamp
notify: whether we notify users new message by email, bool, default FALSE.

Hoods: ID and names of every hood
Primary key: hid
hid, hname: unique id identifies each hood, name of hood
SW & NE: SoutheEast and NorthEast boundary points of hood

Blocks: ID, name and which hood every block belongs to
Primary key: bid
Foreign key: hid refers to hid in Hoods
bid: unique id identifies each block
bname: name of block
hid: hood each block locates in
SW & NE: SoutheEast and NorthEast boundary points of block

UserBlock: Members of each block and joining time
Primary key: uid and ubTime
Foreign key: uid refers to uid in Users; bid refers to bid in Blocks
uid: user id of member
ubTime: time when member joins the block
bid: block member joins
status: Mark this membership is valid or not (True or False, Default True)

BlockApplication: Application of joining a block (temporary)
Primary key: applicant and bid
Foreign key: applicant refers to uid in Users; bid refers to bid in Blocks
applicant: user who apply to join the block
baTime: time when application is submitted
bid: block user apply to join
txt: text applicant can attach with
accepts, decisisons: number of members accepted and decided on the application, default 0
New data can only be inserted after duplicate check

Friends: Friends pairs and the time two users become friends
Primary key: uidA and uidB
Foreign key: uidA refers to uid in Users; uidB refers to uid in Users
uidA & uidB: user id who become friends. uidA < uidB
fTime: time users become friend

FriendApplication: Application for requests to be friends (temporary)
Primary key: applicant and recipient
Foreign key: applicant refers to uid in Users; recipient refers to uid in Users
applicant: user apply for friendsship
recipient: user receive application
txt: text applicant can attach with
faTime: time when application is submitted
decision: recipient accepts or rejects, int (default -1)
New data can only be inserted after duplicate check

Neighbors: User can specify neighbors unilaterally
Primary key: uidA and uidB
Foreign key: uidA refers to uid in Users; uidB refers to uid in Users
uidA & uidB: use A specifies user B as his/her neighbor
nTime: time when user specifies neighbor

Message: Messages sent by users
Primary key: uid, msgid
Foreign key: author refers to uid in Users
msgid: unique id identifies each message thread
author: author of initial message in this thread
rRange: recipient range, limited to: particular(0) / friends(1) / neighbors(2) / block(3) / hood(4), def 4
mtime: time when initial message is put
title & sub &txt: title, subject and text. Photo can be embedded in text with path in special format
coord: co-ordinates where author sends the message, default null

Recipient: Individuals receive the message when rRange is paricular(0)
Primary key: msgid
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
msgid: message thread where the recipient is assigned
uid: user id who is assigned to be recipient

MailBox: Mail box for each user
Primary key: uid and msgid
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
uid: owner if mailbox
msgid: message owner can read and reply
rd: bool variant marking read or not, mark False if not read, True if read, default False

Reply: Replies to message threads
Primary key: msgid, uid, rTime
Foreign key: msgid refers to msgid in Message; uid refers to uid in Users
msgid: message thread the reply belongs to
uid: user who replies
rTime: time when reply is put
txt: content of reply
coord: co-ordinates where replier sends the message, default null

iv. Functional Design Description

This part mainly shows system demand analysis, which is how we design our backend to meet every need.

Function 1: Users should register for the service, specify where they live, post profiles, introduce themselves, upload photos and specify neighbors. Users can choose whether to be notified by email.

Design 1: We designed a Users table to store all kinds of information we need for each user. We use two attributes addr1 and addr2 to represent an address, where addr1 stores street and number, P.O. box, c/o. and addr2 stores apartment, suite, unit, building, floor, etc. Specifically, we design an attribute nRange, which enables users to specify their neighbor range (users should have joined the block of current hood). And there are three available choices for nRange: "same building", "same block" and "same hood". We store these choices in integer format in our database: 0/1/2. As for addresses, we will preprocess them into a certain format in our front-end before inserting. We record and update last-log time (lastLog) every time user logs out. And users can choose whether to be notified by email by setting notify.

Relevant Tables:

Users(<u>uid</u>, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)

Function 2: In the website, there are two levels of locality, hoods and blocks. Both cannot be created by users and are modeled as axis-aligned rectangles that can be defined by two corner points.

Design 2: We designed Hoods and Blocks table to implement this function. For both hoods and blocks, we record a unique id for each data to make them easier to use (hid and bid). We can then acquire full information about given block or hood. Meanwhile, a block is a part of a neighborhood and each block only belongs to one neighborhood. So, we add hid to Blocks. Since blocks and neighborhoods are defined by two corner points, we use SW (southwest) and NE (northeast) corner points to model them.

Relavant Tables:

Hoods(<u>hid</u>, hname, SW, NE)

Blocks(<u>bid</u>, bname, hid, SW, NE)

Function 3: Users can apply to join a block, and they are accepted as a new member if at least three existing members or all members if less than three approve. A user can only be in one block and automatically a member of hood in which the block is located.

Design 3: We designed a BlockApplication table to temporarily store join-block application. When someone submits an application, we store it in BlockApplication table. User can attach some notes in txt in order to get his/her applications approved with higher probability. Every time a user logs in, we scan this table and check whether there is someone trying to join his/her block. If there is one application record whose bid is just the same as his/her block's bid, we push a notification to let him/her decide whether to allow the applicant to join(Yes) or not(No). We add 1 to accepts when someone presses "Yes", and 1 to decisions no matter Yes or No. Every time either of them increases, we check whether the applicant can join the block. If the number of accepts satisfies the requirement, we put the user into UserBlock table, which means he/she is officially a member of the block now. Otherwise, if the number in decisions is equal to the number of members in such block (can be counted in UserBlock) but the accepts does not meet the requirement, we push a rejection notification to the applicant. Besides, system regards applications existing over two weeks also as rejected. After applicant receives either notification, we will delete them from BlockApplication. Every time an applicant submits a request, we check whether there is already one and not expired in BlockApplication. (can be determined by applicant and bid). If there is, we notify the applicant that he/she has already submitted. Since the relationship between hood and block is established in Blocks table, we can join UserBlock and Blocks to deduce which hood a user is automatically a member of.

Relevant Tables:

Blocks(bid, bname, hid, SW, NE)
UserBlock(uid, ubTime, bid, status)
BlockApplication(applicant, bid, baTime, txt, accepts, decisions)

Function 4: Members can specify two types of relationships (friends or neighbors)
Design 4: We've explained the attribute nRange in Users table, which allows users to unilaterally choose his/her neighbors. Thus, we designed table Neighbors for each user (uidA) to store neighbors (uidB) they choose. We also store the time an user specifies neighbor in nTime. For friendship, we designed FriendApplication and Friends tables. They are similar to UserBlock and BlockApplication mentioned in Design 3. One can request friendship and recipient will decide whether to accept. This will be recorded in FriendApplication temporarily, which will be deleted once decision is made. If recipient accepts, a new record will be inserted to Friends. Particularly, we need to point it out that we will check whether applicant and recipient are both in a row from Friends (as uidA and uidB). If it is, we will not initiate the request and immediately notify applicant that they are already friends. Or, if an application is already in FriendApplication (applicant and recipient can identify), we will return to applicant that he/she has already pulled the request. Nothing will change in Friends and FriendApplication under both circumstances.
Relevant Tables:
UserBlock(uid, ubTime, bid, status)
BlockApplication(applicant, bid, baTime, txt, accepts, decisions)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
FriendApplication(applicant, recipient, txt, faTime, decision)

Function 5: People can post, read and reply messages. A user can send message to a person who is a friend or a neighbor, or all of their friends, or entire block, or entire hood they are a member of. Reply can be read and replied by anyone who received the earlier message. Feeds can be separated.
Design 5: We designed Message, Reply and MailBox tables to meet these requirements. We use msgid to identify each thread of messages. We only have the initial message in Message, while all replies are stored in Reply and identified by msgid from Message. And author, mTime, title, sub and rRange are only in Message, because they are redundant to Reply. For rRange, we limit them to predefined choices: particular, friends, neighbor, block and hood. We store these as preset integers: particular (0) / friends(1) / neighbor (2) / block(3) / hood(4)). If a user chooses to send a message to a group (friends, block, etc.), we can query relevant tables (Blocks, Neighbors, UserBlock, Friends, etc.) and deduce who they are specifically, then store the message into MainBox according to uid, msgid and mark rd as False, also we store it into author's and mark read. Or, if a user wants to direct a message to some particular people (by just presenting their names), he shall choose "particular" and we will store all these particular individuals into Recipient table based on msgid. And before we actually act, we will check whether those recipients are his/her friends or neighbors by querying tables: Friends, Neighbors, UserBlock and Blocks. As for replies, if he/she can receive such message in MailBox, then he/she is one of the message's recipients or author (join Message, Reply, Recipient). For MailBox, we check and push notification every time users log in. Every time a new message or reply is put, we mark message thread (msgid in MailBox) as unread for all recipients, unless he/she is author or replier. And we can separate messages into neighbor, friend, block and hood according to rRange (Message, MailBox, UserBlock, Users, Friends). For each thread, we list messages according to mTime or rTime.
Relevant Tables:
Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)

Blocks(bid, bname, hid, SW, NE)
UserBlock(uid, ubTime, bid, status)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)
Recipient(msgid, uid)
MailBox(uid, msgid, rd)
Reply(msgid, uid, rTime, txt, coord)

Fuction 6: System can show only threads with new messages since the last time visited, or profiles of new members, or threads with new messages unread.
Design 6: In Users, we record user's last log-out information (lastLog). We can filter message since lastLog, then query MailBox (msgid) and Message (mTime) based on it. The same thing is with new members since we also store the time when a new member joins (ubTime in UserBlock). As for threads with new message unread, we've described it in Design 5 by using rd in MailBox. Every time a new message or reply comes up, we mark it as unread.
Relevant Tables:
Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)
UserBlock(uid, ubTime, bid, status)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)
MailBox(uid, msgid, rd)

Function 7: Users move to another block
Design 7: If users apply to join another block and get approved, we will not delete the previous data in UserBlock but add new rows (we have ubTime as part of primary key), and mark the status attribute in old row as invalid (False). Since we already treat our messages as Email, users can decide whether to hide previous messages. We can do this by comparing mTime in and ubTime. The reason we choose it this way is that we think there may be some important information in the past messages user might need to look up again in the future. However, since the user is no longer in the previous block, he/she would not be in the recipients relevant to it any longer. Thus, he/she shall not receive any new message threads from previous blocks or hoods. As for friends, we think friendship may last forever, so we won't delete friend information. However, users can hide old friends by comparing fTime and ubTime in Friends and UserBlock. In all, we will keep the past information about messages and friends for users and users can choose to hide this information or not. However, neighbors are unilaterally from the start, so after a user moves to another block and join the new group, he/she should select his/her new neighbors (all old ones will be deleted).
Relevant Tables:
Users(uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)
UserBlock(uid, ubTime, bid, status)
Friends(uidA, uidB, fTime)
Neighbors(uidA, uidB, nTime)
Message(msgid, author, rRange, mtime, title, sub, txt, coord)

b) Create Database

This part is how we build our database named neighborChat

```
DROP DATABASE IF EXISTS neighborChat;
CREATE DATABASE neighborChat;
USE neighborChat;

DROP TABLE IF EXISTS Users;
CREATE TABLE Users (
      uid INT(8) NOT NULL AUTO_INCREMENT,
    uname VARCHAR(50) UNIQUE NOT NULL,
    passwd VARCHAR(128) NOT NULL,
    email VARCHAR(50) NOT NULL,
      fName VARCHAR(50) NOT NULL,
      lName VARCHAR(50) NOT NULL,
      addr1 VARCHAR(50) DEFAULT NULL,
      addr2 VARCHAR(50) DEFAULT NULL,
      intro VARCHAR(50) DEFAULT NULL,
      photo VARCHAR(100) DEFAULT NULL,
    nRange INT(1) DEFAULT 2,
    lastLog TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    notify BOOL NOT NULL DEFAULT FALSE,
      PRIMARY KEY (uid));

DROP TABLE IF EXISTS Hoods;
CREATE TABLE Hoods (
      hid INT(8) NOT NULL AUTO_INCREMENT,
      hname VARCHAR(50) NOT NULL,
      SW VARCHAR(50) NOT NULL,
      NE VARCHAR(50) NOT NULL,
      PRIMARY KEY (hid));

DROP TABLE IF EXISTS Blocks;
CREATE TABLE Blocks (
      bid INT(8) NOT NULL AUTO_INCREMENT,
      bname VARCHAR(50) NOT NULL,
      hid INT(8) NOT NULL,
      SW VARCHAR(50) NOT NULL,
      NE VARCHAR(50) NOT NULL,
      PRIMARY KEY (bid),
      FOREIGN KEY (hid) REFERENCES Hoods (hid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS UserBlock;
CREATE TABLE UserBlock (
      uid INT(8) NOT NULL,
    ubTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    bid INT(8) NOT NULL,
    status BOOL DEFAULT TRUE,
      PRIMARY KEY (uid, ubTime),
      FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (bid) REFERENCES Blocks (bid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS BlockApplication;
CREATE TABLE BlockApplication (
      applicant INT(8) NOT NULL,
      baTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
      bid INT(8) NOT NULL,
      txt VARCHAR(100) DEFAULT NULL,
      accepts INT DEFAULT 0,
      decisions INT DEFAULT 0,
      PRIMARY KEY (applicant, bid),
      FOREIGN KEY (applicant) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (bid) REFERENCES Blocks (bid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Friends;
CREATE TABLE Friends (
      uidA INT(8) NOT NULL,
```

```
    uidB INT(8) NOT NULL,
    fTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (uidA, uidB),
    FOREIGN KEY (uidA) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uidB) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS FriendApplication;
CREATE TABLE FriendApplication (
    applicant INT(8) NOT NULL,
    recipient INT(8) NOT NULL,
  txt VARCHAR (100) DEFAULT NULL,
  faTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  decision INT(1) DEFAULT -1,
    PRIMARY KEY (applicant, recipient),
    FOREIGN KEY (applicant) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (recipient) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Neighbors;
CREATE TABLE Neighbors (
    uidA INT(8) NOT NULL,
    uidB INT(8) NOT NULL,
    nTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (uidA, uidB),
    FOREIGN KEY (uidA) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uidB) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Message;
CREATE TABLE Message (
    msgid INT(8) NOT NULL AUTO_INCREMENT,
    author INT(8) NOT NULL,
    rRange INT(8) DEFAULT 4,
    mtime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title VARCHAR(50) NOT NULL,
    sub VARCHAR(50) DEFAULT NULL,
    txt VARCHAR(1000) NOT NULL,
    coord VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (msgid),
    FOREIGN KEY (author) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Recipient;
CREATE TABLE Recipient (
    msgid INT(8) NOT NULL,
    uid INT(8) NOT NULL,
    PRIMARY KEY (msgid),
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS MailBox;
CREATE TABLE MailBox (
    uid INT(8) NOT NULL,
    msgid INT(8) NOT NULL,
    rd BOOL NOT NULL DEFAULT FALSE,
    PRIMARY KEY (uid, msgid),
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE);

DROP TABLE IF EXISTS Reply;
CREATE TABLE Reply (
    msgid INT(8) NOT NULL,
    uid INT(8) NOT NULL,
    rTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    txt VARCHAR(1000) NOT NULL,
    coord VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (msgid, uid, rTime),
    FOREIGN KEY (msgid) REFERENCES Message (msgid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (uid) REFERENCES Users (uid) ON DELETE CASCADE ON UPDATE CASCADE);
```

c) SQL Queries

This part are SQL queries for given tasks

i. Join

-- Users sign up

```
INSERT INTO Users VALUES (1, "user01", "12345678", "test@gmail.com", "Justin", "Bieber", NULL, NULL, NULL, "../static/plugins/images/users/default.png", FALSE, "2019-01-01 12:00:00", FALSE);
```

-- Users apply to become members of a block

```
INSERT INTO BlockApplication VALUES (6, "2019-01-01 12:00:02", 2, "I am your new neighbor", 0, 0);
```

-- Users create profiles

```
UPDATE Users SET passwd = "12345678", email = "test@gmail.com", fName = "Justin", lName = "Bieber", addr1 = "243 Gold Street, Brooklyn", addr2 = "Apt 0001", intro = "LOL", photo = "../static/plugins/images/users/default.png", notify = TRUE WHERE uid = 1;
```

-- Users edit profiles

```
UPDATE Users SET passwd = "12345678", email = "test@gmail.com", fName = "Justin", lName = "Bieber", addr1 = "343 Gold Street, Brooklyn", addr2 = "Apt 4001", intro = "Hello World!", photo = "../static/plugins/images/users/1.png ", notify = TRUE WHERE uid = 1;
```

ii. Content Posting

-- User starts a thread by posting an initial message

```
INSERT INTO Message VALUES (6, 9, 0, "2019-01-01 12:00:04", "Hi", "Life", "I love you", "(40.7657, -73.9761)");
```

-- User replies to a message

```
INSERT INTO Reply VALUES (6, 10, "2019-01-01 13:00:04", "I love you too", "");
```

iii. Friendship

-- User applies for friendship

```
INSERT INTO FriendApplication VALUES (2, 3, "I wanna be your friend", "2019-01-01 12:00:03", -1);
```

-- User accepts friend request

```
DELETE FROM FriendApplication WHERE applicant = 2 AND recipient = 3;
INSERT INTO Friends VALUES (2, 3, "2019-11-29 12:00:03");
```

-- User adds new neighbor

```
INSERT INTO Neighbors VALUES (3, 1, "2019-01-01 12:00:03");
```

-- All current friends to user whose uid = 2

```
SELECT * FROM Friends WHERE uidA = 2 OR uidB = 2;
```

-- All current neighbors to uid = 3

```
SELECT * FROM Neighbors WHERE uidA = 3;
```

iv. Browse and Search Messages

-- List all threads in a user(9)'s block feed that have new messages since the last time the user accessed

```
SELECT * FROM Message
WHERE rRange = 3 AND mtime >= (SELECT lastLog FROM Users WHERE uid = 9)
AND author in (SELECT uid FROM UserBlock WHERE bid = (SELECT bid FROM UserBlock WHERE uid = 9));
```

-- List all threads in user(2)’s friend feed that have unread messages

```
SELECT * FROM Message NATURAL JOIN MailBox WHERE uid = 2 AND rd = FALSE AND rRange = 1;
```

-- List all messages containing the words “bicycle accident” across all feeds that user(8) can access

```
SELECT * FROM Mailbox NATURAL JOIN Message
WHERE uid = 8 AND (title LIKE "%bicycle accident%" OR sub LIKE "%bicycle accident%" OR txt LIKE "%bicycle accident%");
```

d) Sample Data Test

i. Insert Sample Data

This part we insert sample data in order to test our database

Users (uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify)

nRange: same building(0) / block(1) / hood(2)

-- Live in the same building, different nRange: uid 1 – 3; uid 10 - 11

-- Live in the same block, different nRange: uid 1 – 3; uid 4 – 6; uid 7; uid 8 – 12

-- Live in the same hood, different nRange: uid 1 – 6; uid 7 – 12

uid	uname	passwd	email	fName	lName	addr1	addr2	intro	photo	nRange	lastLog	notify
1	user01	12345678	test@gmail.com	Justin	Bieber	343 Gold Street, Brooklyn	Apt 4001	Hello World!	../static/plugins/images/users/1.png	0	2019-01-01 12:00:00	1
2	user02	12345678	test@gmail.com	Donald	Trump	343 Gold Street, Brooklyn	Apt 4002	Hello World!	../static/plugins/images/users/2.png	1	2019-01-01 12:00:00	1
3	user03	12345678	test@gmail.com	Chris	Martin	343 Gold Street, Brooklyn	Apt 4201	Hello World!	../static/plugins/images/users/3.png	2	2019-01-01 12:00:00	1
4	user04	12345678	test@gmail.com	Lady	Gaga	270 Jay Street, Brooklyn		Hello World!	../static/plugins/images/users/4.png	0	2019-01-01 12:00:00	1
5	user05	12345678	test@gmail.com	Anne	Hathaway	320 Jay Street, Brooklyn		Hello World!	../static/plugins/images/users/5.png	1	2019-01-01 12:00:00	1
6	user06	12345678	test@gmail.com	Leonardo	Dicaprio	370 Jay Street, Brooklyn		Hello World!	../static/plugins/images/users/6.png	2	2019-01-01 12:00:00	1
7	user07	12345678	test@gmail.com	Billie	Ellish	500 5th Avenue, New York		Hello World!	../static/plugins/images/users/7.png	2	2019-01-01 12:00:00	1
8	user08	12345678	test@gmail.com	James	Bond	1100 6th Avenue, New York		Hello World!	../static/plugins/images/users/8.png	1	2019-01-01 12:00:00	1
9	user09	12345678	test@gmail.com	Adam	Levine	1166 6th Avenue, New York		Hello World!	../static/plugins/images/users/9.png	2	2019-01-01 12:00:00	1
10	user10	12345678	test@gmail.com	Bruno	Mars	1167 6th Avenue, New York		Hello World!	../static/plugins/images/users/10.png	0	2019-01-01 12:00:00	1
11	user11	12345678	test@gmail.com	Scarlett	Johansson	1167 6th Avenue, New York		Hello World!	../static/plugins/images/users/11.png	1	2019-01-01 12:00:00	1
12	user12	12345678	test@gmail.com	Robert	Downey	1170 6th Avenue, New York		Hello World!	../static/plugins/images/users/12.png	2	2019-01-01 12:00:00	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Hoods (hid, hname, SW, NE)

-- Live in different hoods: hid 1 - 3

hid	hname	SW	NE
1	Downtown Brooklyn	(40.6902, -73.9943)	(40.7059, -73.9809)
2	Midtown Manhattan	(40.7477, -73.9929)	(40.7647, -73.9739)
3	Uptown Bronx	(40.8113, -73.9315)	(40.8830, -73.7945)
NULL	NULL	NULL	NULL

Blocks (bid, bname, hid, SW, NE)

-- Live in different blocks: bid 1 - 4

bid	bname	hid	SW	NE
1	Jay Street	1	(40.6962, -73.9872)	(40.6998, -73.9868)
2	Gold Street	1	(40.6922, -73.9834)	(40.7056, -73.9823)
3	5th Avenue From 34th Street to 59th Street	2	(40.7485, -73.9846)	(40.7644, -73.9730)
4	6th Avenue From 34th Street to 59th Street	2	(40.7498, -73.9878)	(40.7657, -73.9761)
NULL	NULL	NULL	NULL	NULL

UserBlock (uid, ubTime, bid, status)

-- For block 1, all users (uid 1-3) in this block have joined the block group (3 accepts required)

-- For block 2, two out of three users (uid 4-5) have already joined (all members accept required)

-- For block 3, a user (uid 7) is the first person to join the block (applicant already automatically joined)

-- For block 4, four out of five users (uid 8-11) have already joined (3 acceptes required)

uid	ubTime	bid	status
1	2019-01-01 12:00:01	1	1
2	2019-01-01 12:00:01	1	1
3	2019-01-01 12:00:01	1	1
4	2019-01-01 12:00:01	2	1
5	2019-01-01 12:00:01	2	1
7	2019-01-01 12:00:01	3	1
8	2019-01-01 12:00:01	4	1
9	2019-01-01 12:00:01	4	1
10	2019-01-01 12:00:01	4	1
11	2019-01-01 12:00:01	4	1
NULL	NULL	NULL	NULL

BlockApplication(applicant, bid, baTime, txt, accepts, decisions)

-- For block 2, applicant (6) applies to join. Since block members <= 3, all members agreement required

-- For block 3, block member was 0, applicant 7 automatically joined, so there is no record here

-- For block 4, , applicant (12) applies to join. Since block member > 3, only 3 accepts required

applicant	baTime	bid	txt	accepts	decisions
6	2019-01-01 12:00:02	2	I am your new neighbor	0	0
12	2019-01-01 12:00:02	4	I am your new neighbor	0	0
NULL	NULL	NULL	NULL	NULL	NULL

Friends (uidA, uidB, fTime)

-- Two people already in the same block group: uid 1&2; uid 4&5

-- Two people live in the same hood, but not the same block: uid 7&8, uid 7&10

-- One moves to a new hood maintaining old friends and making new friends: uid 2&11

uidA	uidB	fTime
1	2	2019-01-01 12:00:03
2	11	2019-02-01 12:00:03
4	5	2019-01-01 12:00:03
7	8	2019-01-01 12:00:03
7	10	2019-01-01 12:00:03
NULL	NULL	NULL

FriendApplication (applicant, recipient, txt, faTime, decision)

-- Two people already in the same block group: applicant 2, 8

-- Two people live in the same hood, but not the same block: applicant 1, 7

applicant	recipient	txt	faTime	decision
1	4	I wanna be your friend	2019-01-01 12:00:03	-1
2	3	I wanna be your friend	2019-01-01 12:00:03	-1
7	11	I wanna be your friend	2019-01-01 12:00:03	-1
8	9	I wanna be your friend	2019-01-01 12:00:03	-1
NULL	NULL	NULL	NULL	NULL

Neighbors (uidA, uidB, nTime)

-- Users select their neighbor range unilaterally (referenced to nRange in Users table)

uidA	uidB	nTime
1	2	2019-01-01 12:00:03
1	3	2019-01-01 12:00:03
2	1	2019-01-01 12:00:03
2	3	2019-01-01 12:00:03
3	1	2019-01-01 12:00:03
3	2	2019-01-01 12:00:03
3	4	2019-01-01 12:00:03
3	5	2019-01-01 12:00:03
5	4	2019-01-01 12:00:03
7	8	2019-01-01 12:00:03
7	9	2019-01-01 12:00:03
7	10	2019-01-01 12:00:03
7	11	2019-01-01 12:00:03
8	9	2019-01-01 12:00:03
8	10	2019-01-01 12:00:03
8	11	2019-01-01 12:00:03
9	7	2019-01-01 12:00:03
9	8	2019-01-01 12:00:03
9	10	2019-01-01 12:00:03
9	11	2019-01-01 12:00:03
10	11	2019-01-01 12:00:03
11	8	2019-01-01 12:00:03
11	9	2019-01-01 12:00:03
11	10	2019-01-01 12:00:03
NULL	NULL	NULL

Message (msgid, author, rRange, mtime, title, sub, txt, coord)

-- Someone sends message to friends: msgid 1

-- Someone sends message to neighbors, and coordinates is null: msgid 2

-- Someone sends message to block: msgid 3

-- Someone sends message to hood, and coordinates is null: msgid 4

-- Someone sends message to particular person from friends: msgid 5

-- Someone sends message to particular person from neighbors: msgid 6- 7

-- Specifically, for bicycle accident: msgid 8 - 10

---- Keyword in title: msgid 8 ---- Keyword in subject: msgid 9 ---- Keyword in text: msgid 10

msgid	author	rRange	mtime	title	sub	txt	coord
1	1	1	2019-01-01 12:00:04	LOL	Life	I am happy	(40.6962, -73.9872)
2	5	2	2019-01-01 12:00:04	???	Life	Stop using my Wi-Fi	
3	8	3	2019-01-01 12:00:04	Help	Work	Can someone help me with my school work?	(40.7498, -73.9878)
4	7	4	2019-01-01 12:00:04	Vote time	Food	Which one do you prefer, medium rare or mediu…	
5	7	0	2019-01-01 12:00:04	Hello	Life	How's your weekend?	(40.7657, -73.9761)
6	9	0	2019-01-01 12:00:04	Hi	Life	I love you	(40.7657, -73.9761)
7	2	0	2019-01-01 12:00:04	Invitation	Life	Have dinner with me?	(40.7056, -73.9823)
8	9	2	2019-01-01 12:00:04	Bicycle Accident	Emergency	Somebody is hit by a bicycle in the block	(40.7498, -73.9878)
9	10	3	2019-01-01 12:00:04	Terrible!	Bicycle Accident	I was hit by a bicycle...	(40.7498, -73.9878)
10	11	4	2019-01-01 12:00:04	Accident Report	Accident	There is a bicycle accident in the block!	(40.7498, -73.9878)
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Recipient (msgid, uid)

-- Recipient test data based on test data in Message: msgid 5 - 7

msgid	uid
7	3
5	8
6	10
NULL	NULL

Reply (msgid, uid, rTime, txt, coord)

-- Reply to particular message: msgid 6

-- Reply to group message: msgid 4

msgid	uid	rTime	txt	coord
4	8	2019-01-01 12:00:06	Medium Rare	
4	9	2019-01-01 12:00:06	Medium	(40.7583, -73.9815)
4	10	2019-01-01 12:00:06	Medium Rare	
4	11	2019-01-01 12:00:06	Medium	(40.7630, -73.9781)
6	10	2019-01-01 12:00:04	I love you too	
NULL	NULL	NULL	NULL	NULL

MailBox (uid, msgid, rd)

-- Based on three tables above(Message, Recipient, Reply), we can draw the following table, which can be used to deduce MailBox.

msgid	nRange	author	recipient(s)
1	Friend	1	2
2	Neighbor	5	4
3	Block	8	9, 10, 11
4	Hood	7	7, 8, 9, 10, 11
5	Particular	7	8
6	Particular	9	9, 10
7	Particular	2	3
8	Neighbor	9	7, 8, 10, 11
9	Block	10	8, 9, 11
10	Hood	11	7, 8, 9, 10

-- General test data based on test data in Message: msgid 1 - 3

-- They need to reply, so they must have read the message: uid 8 - 11, msgid 4

-- After being replied, author of message thread will have his/her mailbox updated: uid 7, msgid 4

-- Specifically for bicycle accident: uid 7 - 11, msgid 8 - 10

msgid	uid	rd
1	2	0
2	4	1
3	9	0
3	10	1
3	11	1
4	7	0
4	8	1
4	9	1
4	10	1
4	11	1
5	8	0
6	9	0
6	10	1

7	3	1
8	7	0
8	8	0
8	10	0
8	11	0
9	8	0
9	9	0
9	11	0
10	7	0
10	8	0
10	9	0
10	10	0
NULL	NULL	NULL

ii. Test Queries

This part we test SQL queries written in c) with sample data from d) i

Join

-- Users sign up

uid	uname	passwd	email	fName	lName	addr1	addr2	intro	photo	nRange	lastLog	notify
1	user01	12345678	test@gmail.com	Justin	Bieber	NULL	NULL	NULL	../static/plugins/images/users/default.png	0	2019-01-01 12:00:00	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- Users apply to become members of a block

applicant	baTime	bid	txt	accepts	decisions
6	2019-01-01 12:00:02	2	I am your new neighbor	0	0
NULL	NULL	NULL	NULL	NULL	NULL

-- Users create profiles

uid	uname	passwd	email	fName	lName	addr1	addr2	intro	photo	nRange	lastLog	notify
1	user01	12345678	test@gmail.com	Justin	Bieber	243 Gold Street, Brooklyn	Apt 0001	LOL	../static/plugins/images/users/default.png	0	2019-01-01 12:00:00	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- Users edit profiles

uid	uname	passwd	email	fName	lName	addr1	addr2	intro	photo	nRange	lastLog	notify
1	user01	12345678	test@gmail.com	Justin	Bieber	343 Gold Street, Brooklyn	Apt 4001	Hello World!	../static/plugins/images/users/1.png	0	2019-01-01 12:00:00	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Content Posting

-- User starts a thread by posting an initial message

msgid	author	rRange	mtime	title	sub	txt	coord
6	9	0	2019-01-01 12:00:04	Hi	Life	I love you	(40.7657, -73.9761)
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- User replies to a message

msgid	uid	rTime	txt	coord
6	10	2019-01-01 13:00:04	I love you too	
NULL	NULL	NULL	NULL	NULL

Friendship

-- User applies for friendship

applicant	recipient	txt	faTime	decision
2	3	I wanna be your friend	2019-01-01 12:00:03	NULL
NULL	NULL	NULL	NULL	NULL

-- User accepts friend request

uidA	uidB	fTime
2	3	2019-11-29 12:00:03
NULL	NULL	NULL

-- User adds new neighbor

uidA	uidB	nTime
3	1	2019-01-01 12:00:03
NULL	NULL	NULL

-- All current friends to user whose uid = 2

uidA	uidB	fTime
1	2	2019-01-01 12:00:03
2	11	2019-02-01 12:00:03
NULL	NULL	NULL

-- All current neighbors to uid = 3

uidA	uidB	nTime
3	1	2019-01-01 12:00:03
3	2	2019-01-01 12:00:03
3	4	2019-01-01 12:00:03
3	5	2019-01-01 12:00:03
NULL	NULL	NULL

Browse And Search Messages

-- List all threads in a user(9)'s block feed that have new messages since the last time the user accessed

msgid	author	rRange	mtime	title	sub	txt	coord
3	8	3	2019-01-01 12:00:04	Help	Work	Can someone help me with my school work?	(40.7498, -73.9878)
9	10	3	2019-01-01 12:00:04	Terrible!	Bicycle Accident	I was hit by a bicycle...	(40.7498, -73.9878)
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- List all threads in user(2)’s friend feed that have unread messages

msgid	author	rRange	mtime	title	sub	txt	coord	uid	rd
1	1	1	2019-01-01 12:00:04	LOL	Life	I am happy	(40.6962, -73.9872)	2	0

-- List all messages containing the words “bicycle accident” across all feeds that user(8) can access

msgid	uid	rd	author	rRange	mtime	title	sub	txt	coord
8	8	0	9	2	2019-01-01 12:00:04	Bicycle Accident	Emergency	Somebody is hit by a bicycle in the block	(40.7498, -73.9878)
9	8	0	10	3	2019-01-01 12:00:04	Terrible!	Bicycle Accident	I was hit by a bicycle...	(40.7498, -73.9878)
10	8	0	11	4	2019-01-01 12:00:04	Accident Report	Accident	There is a bicycle accident in the block!	(40.7498, -73.9878)

Part II - Web based User Interface

a) Revising Design

This part we improve our design in Project 1 by fixing some bugs or adding some features

- add attribute decision to FriendApplication

We could not notify applicant that he/she is rejected by recipient before. Now we can notify him/her by checking the decision (-1: not made, 0: rejected, 1: accepted). We now make changes to Friends and FriendApplication after we make sure applicant is well notified.

- add attribute status to UserBlock

We can only know which block a user is in currently by comparing timestamp before, which makes it impossible for a user once in a block return solo (not in any block). Now we add this new attribute so that user who once in a block can now quit and stay solo.

- add attribute notify to Users

We cannot notify users new message by email before. Now we can let them choose whether we do so.

- Fix some Functional Design Description for Message Part.

- Modify BlockApplication, cancelling primary key baTime and adding primary key bid.

With this update, user can apply multi blocks, and accepted by multi blocks. Then, user can decide which block to join. And each block application can only exist one in temporary application table (will be deleted after applying over two weekends)

- add attribute email to Users

We missed it in many places for the first part, now we add it back.

- Modify passwd, changing Users to 128 bytes

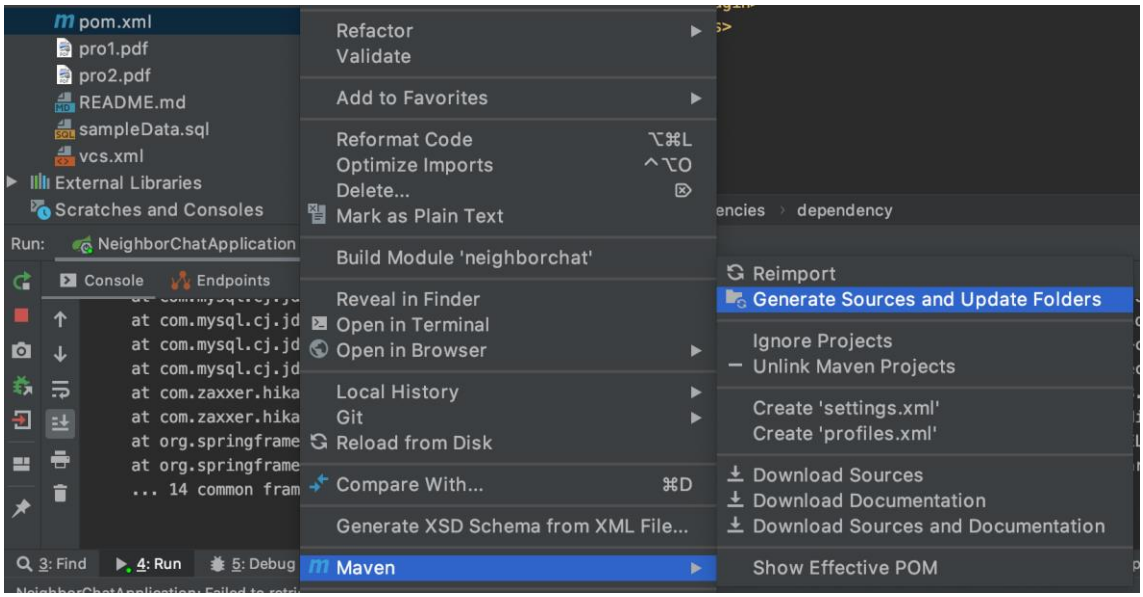
In order to store password with SHA-256 encrypted, now we extend it to 128 bytes.

b) Project Deployment

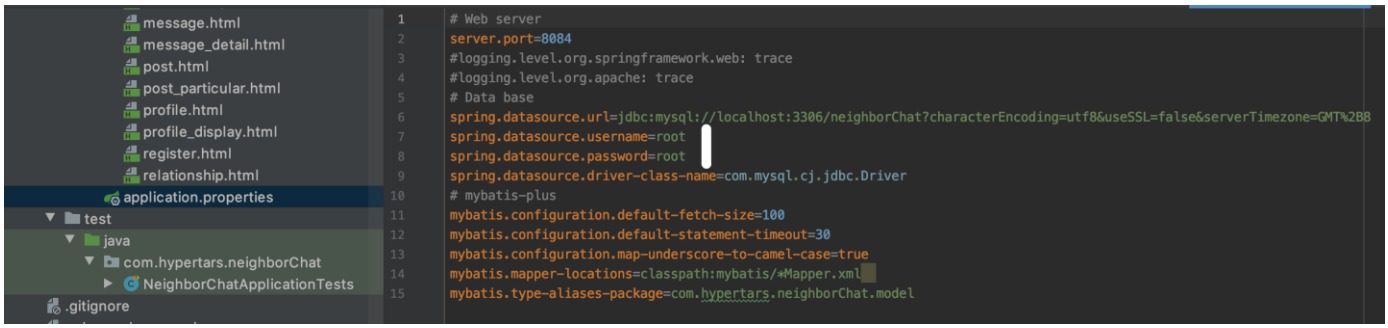
This part is the overview of our project framework and guidance of use.

We build the project with
SpringBoot (Java Servlet Framework),
Maven (project management),
Dubbo (framework),
MySQL (DB),
DAO(DB connection),
MyBatis (DB connection, **SQL inject protection**, **Transaction**),
Apache (local server, **SHA-256 encryption**),
Tomcat (runtime server),
Google API (map use),
BootStrap (front-end library)
Cookie (**user authentication**),
Jsoup (**XSS protection**),
and Intellij IDEA (IDE).

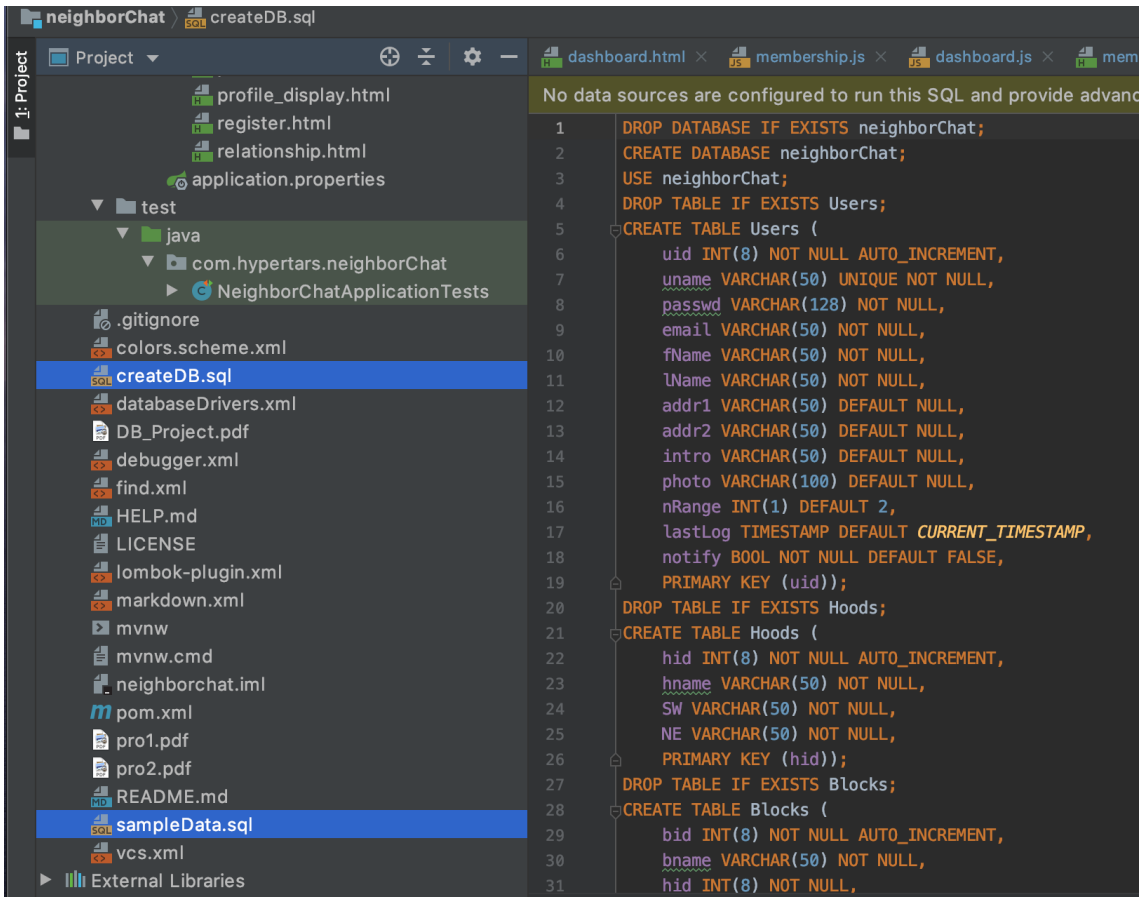
To deploy and test our project, please install Intellij IDEA and import the whole project folder, set Java Runtime Environment, and use Maven (right click pom.xml) to download and construct relevant dependencies as follows.



Then, configure database in application.properties.



Database creation and sample data are as follows. (MySQL)



And this is the file structure of our project:

neighborChat/src/main/java/com/hypertars/neighborChat

- /dao: define SQL execution, interacting with mybatis, interface for DB connection
- /enums: define error code
- /exception: define exception
- /filter: XSS filter to protect our project from XSS
- /model: define Models for each table (attributes)
- /service: define Service for operations, specify multi database executions
- /utils: utilizers
- /web: define Controller to interact with front-end and user authentication

neighborChat/src/main/java/resources

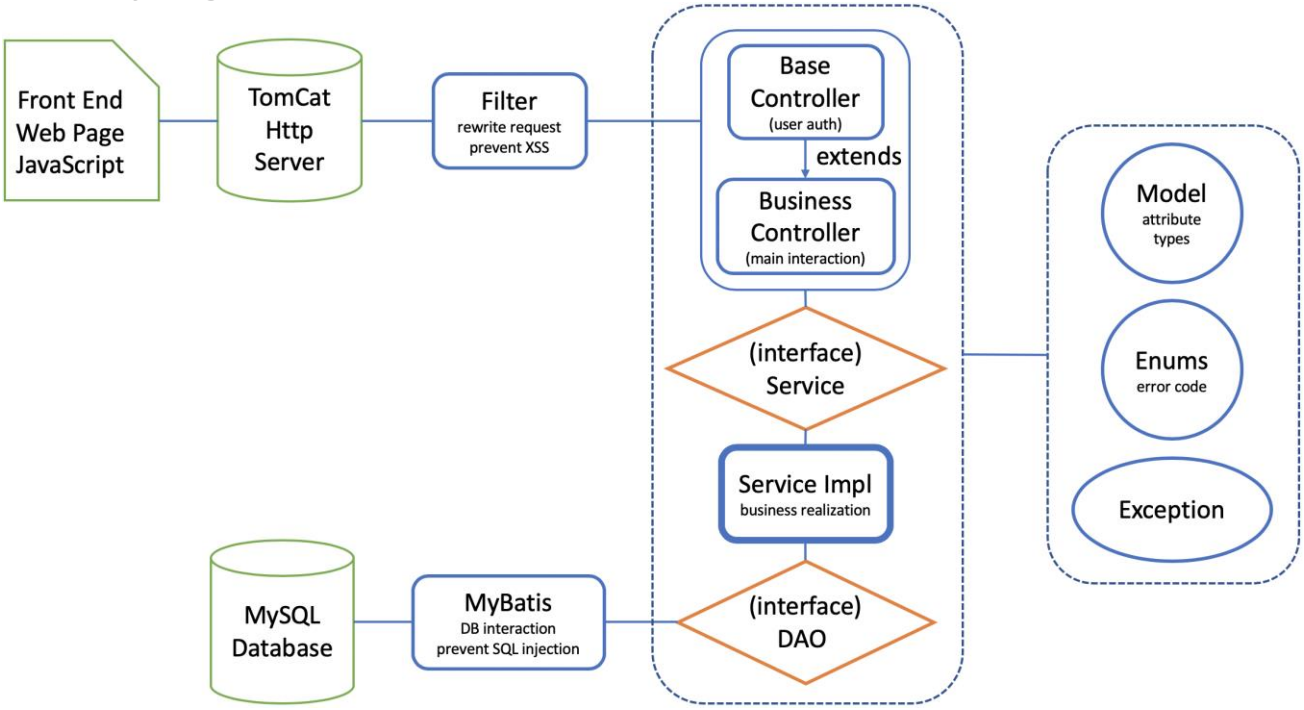
- /mybatis: prepared SQLs for each table, protected from SQL injection
- /static: static resources like css and js
- /templates: dynamic website (html)

c) Back-end Design (Service Design)

This part mainly describes the structure of our back-end

i. Overview

Overview of background



ii. Service & Controller: Business Design

Because of the space limit, we can only describe main business (Controller and Service) in a rough

UserAccount:

Relevant tables: Users

register user info:

- check whether uName is already in use
- If not, add user to Users

user log In & auth:

- check whether uName matches password
- Save user cookie (session) to session map
- remove user cookie if log out

user forget password:

- authentificate user with uName and Email
- If true, allow user to modify password

user Info

- update basic info
- update last log
- query user info by uid

Load Data (schedule scanning)

Relevant tables: All tables

load users

- personal info
- friend list

    - neighbor list
    - same building list
    - stranger list (not friend or neighbor but same block or hood)
    - all users in current block
    - all users in current hood

  load membership
    - current block and block info
    - current hood and hood info

  load message
    - load all messages, classifying by from particular / friend / neighbor / block / hood

Membership

Relevant tables: UserBlock & BlockApplication & Blocks & Hoods & Users

  query all membership in history

  apply for new block (membership)
    - load all blocks available
    - check whether exists in such block
    - check whether exists such block application

  make application decision as recipient
    - load all block application for his/her block
    - accept application
    - reject application

  exit current block
    - set user block inactive
    - delete all neighbors

Relationship

Relevant tables: FriendApplication & Friends & Neighbor & Blocks & Hoods & UserBlock & Users

  query all friends / neighbors / strangers nearby, classifying them with same building, block, hood

  apply for new friend
    - friend list applicable (strangers nearby)
    - check whether friend already exists
    - check whether friend application exists
    - add friend application

  friend application
    - list all friend applications as applicant
    - list all friend applications as recipient
    - accept friend application
    - reject friend application

  add new neighbor

  delete friend / neighbor

Message

Relevant tables: Message & MailBox & Reply & Friends & Neighbors & Recipient & Users

  query all messages and replies
    - mark with read / unread
    - range from particular / friends / neighbors / all block / all hood

write new message

- load possible recipient list for particular message
- add new message to Message
- add new message to MailBox

write new reply

- message threads that you can see are what you can reaply to
- add new reply
- set message in other's MailBox unread

Notification (schedule scanning)

Relevant tables: All Tables

notify new block application from current block as recipient

- can accept or reject

notify accepted block application

- check if any application satisfies accept condition
- condition: block member >= 3: 3 / 0 < block member < 3 : all / block member = 0
- if satisfies, notify and add user to UserBlock, then delete block application
- add user to UserBlock: if new, just add; if old: set old inactive and delete all neighbors

notify rejected block application

- check if any application satisfies reject condition
- condition: block member = reject votes / application exists over 14 days
- if satisfies, notify and delete block application

notify new friend application

- can accept or reject

notify accepted friend application

- add friend pair to Friends table
- delete friend application

notify rejected friend application

- delete friend application

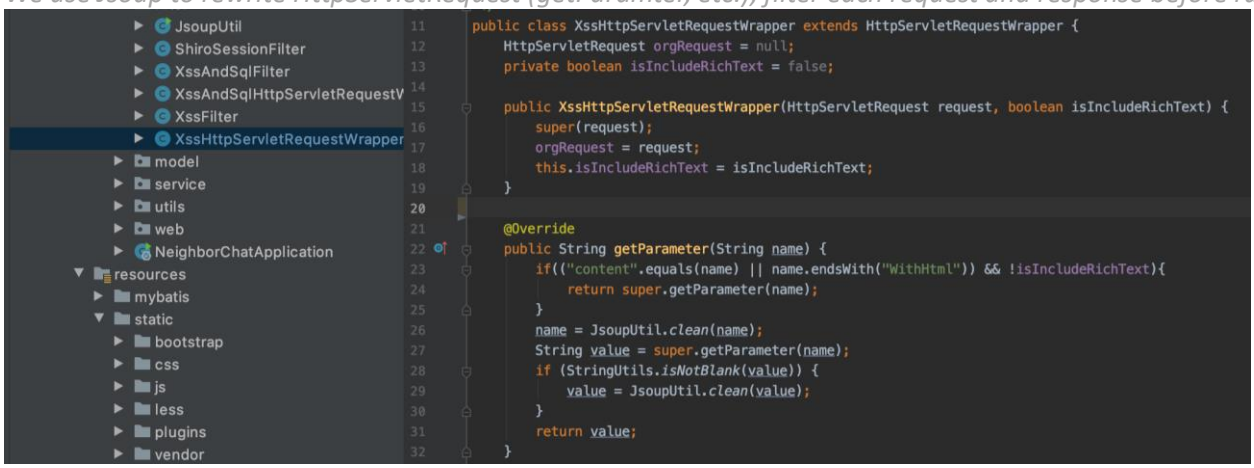
notify new message

notify new reply

update last notified

iii. Filter: XSS Protection

We use Jsoup to rewrite HttpServletRequest (getParamter, etc.), filter each request and response before running



iv. MyBatis: DB connection & SQL Injection Protection

We use MyBatis to precompile our SQLs, protecting our system from SQL injection.

```
Project
  XssHttpServletRequestWrapper
  model
  service
  utils
  web
  NeighborChatApplication
resources
  mybatis
    BlockApplicationMapper.xml
    BlocksMapper.xml
    FriendApplicationMapper.xml
    FriendsMapper.xml
    HoodsMapper.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hypertars.neighborChat.dao.UsersDAO">

    <select id="getUserByUid" resultType="com.hypertars.neighborChat.model.Users" parameterType="java.lang.Integer">
        SELECT uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify
        FROM Users WHERE uid = #{uid}
    </select>

    <select id="getUserByUName" resultType="com.hypertars.neighborChat.model.Users" parameterType="java.lang.String">
        SELECT uid, uname, passwd, email, fName, lName, addr1, addr2, intro, photo, nRange, lastLog, notify
        FROM Users WHERE uname = #{uname}
    </select>
```

v. Feature: SHA-256 Encryption

We use Apache.common-codec to realize password encryption (SHA-256). We never store real password.

```
private static String getSHA256(String str){
    MessageDigest messageDigest;
    String encodeStr = "";
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
        byte[] hash = messageDigest.digest(str.getBytes( charsetName: "UTF-8"));
        encodeStr = Hex.encodeHexString(hash);
    } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    System.out.println(encodeStr);
    return encodeStr;
}
```

uid	uname	passwd
1	user01	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
2	user02	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
3	user03	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
4	user04	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
5	user05	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
6	user06	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
7	user07	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
8	user08	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
9	user09	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
10	user10	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
11	user11	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
12	user12	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...
13	user13	ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532...

vi. Feature: Transaction

We use Spring Boot Annotation to make some essential functions transactional in case multi users operate.

```
@Transactional
@RequestMapping(value = "addMessage", produces = "text/script;charset=UTF-8")
public String addMessage(HttpServletRequest request, String callback) {
    NBCResult<Object> result = new NBCResult<>();
    result = protectController(request,  modelMap: null, new NBCLogicCallBack() {
        @Override
```

vii. Feature: User Auth

We use cookie to do user authentication. For each request, we verify who it belongs to.

```
private void verifyLogin(HttpServletRequest request) {
    try {
        // retrieve Cookie in request header
        String session = "";
        for (Cookie cookie: request.getCookies()) {
            if (StringUtils.equal(USER_COOKIE, cookie.getName())) {
                session = cookie.getValue();
                break;
            }
        }
        AssertUtils.stringNotEmpty(session);
        // retrieve user in session
        Users user = useraccountService.getUserBySession(session);
        AssertUtils.assertNotNull(user);
        loginUsers.set(user);
    } catch (Exception e) {
        throw new NBCException("User not logged in, redirect to log in page...", NBCResultCodeEnum.USER_NOT_LOGIN_IN);
    }
}
```

d) Front-end Design

This part mainly describes the structure of our front-end

Register.html

- Users create an account with their username, first name, last name, email, password.

Login.html

- Users log in the website with their username and password.

Dashboard.html

- Count how many new messages are there in block / hood feed since the last time user logs out
- Display all new messages in Particular, Friend, Neighbor, Block and Hood feeds.

Profile.html

- Display information (including password, address, etc.) and individual settings (including neighbor range's choice, whether to receive notification by email) of current logged-in user.
- User can update this information.

Profile_display.html

- Display only part of information (hiding attributes such as password, etc.) of other users.

Membership.html

- If user hasn't joined a block, he/she can apply to join a block from available blocks list.
- Otherwise, he/she can query current block' s information (ID, name, hood the block belongs to) and query all block members and all hood members. He/she can approve/reject a pending block application from other applicants. He/she can even leave current block.

Relationship.html

- List all friends and neighbors of current user.
- List all pending friend applications.
- List all nearby strangers, users can apply to make friends with them or set them as neighbors.

Post.html

- Current user can post a message to a particular person / all friends / all neighbors / all block members / all hood members.

Message.html

- Serve as a mailbox of current user, which store all messages current user received and posted
- User can search certain message for convenience.

Message_detail.html

- Display the detailed information of each message, including author, title, text, etc.
- Users(including the author) can reply to the message.

e) Project Test & Use

This part mainly describes how we test and use our system

i. Test Case

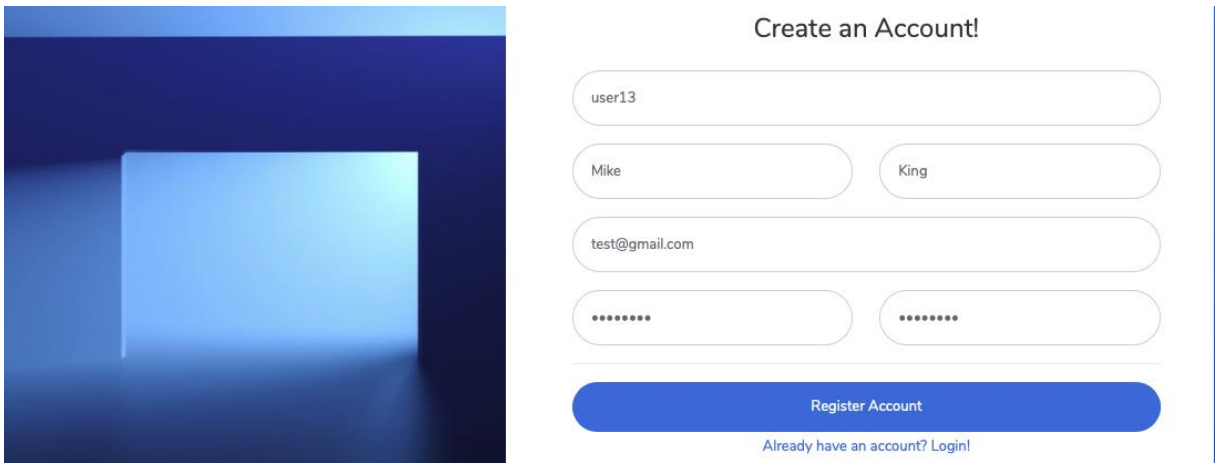
This part mainly describes how we test and use our system

login	1. try to access page without log in, return failure
	2. try to log in with wrong password, return failure
	3. try to log in with right password, return success
	4. logout, clean cookie, try to access page, return failure
	5. register, check whether username is already taken
	6. register, return success, jump to main page logged in
Main & Profile	1. load messages from Block and Hood
	2. load messages from different sources
	3. modify personal information
Membership	1. load messages from Block and Hood
	2. load members from Block and Hood
	3. load all Blocks available to join, try to join and alert application exists if repeat
	4. load Block Applications as applicant and recipient, try to accept and reject
	5. exit current Block, alert if not in any Block
Relationship	1. load Friends list
	2. look friend's or neighbor's or stranger's personal information
	3. load Neighbors list
	4. load Users living in same building
	5. load Strangers can apply for as Friends / Neighbors
	6. apply Friendship，alert if exists
	7. load Friend Application lists
	8. accept Friend Application
	9. reject Friend Application
	10. add Neighbors，alert if exist
	11. delete Neighbors，alert if not exist
	12. delete Friends，alert if not exist
Message	1. load Message
	2. open Message Thread to see complete content and Reply attached to it
	3. search Message based on keyword
	4. write new Message (with location and particular recipient or range)
	5. write new Reply
notify (based on last notification)	1. notify new Block Application (as recipient)
	2. notify if Block Application passed (>3 or =all or new Block), add User Block, delete BA
	3. notify if Block Application rejected (reject=all), delete Block Application
	4. notify new Friend Application (as recipient)
	5. notify Friend Application passed，add Friends，delete Friend Application
	6. notify Friend Application rejected，delete Friend Application
	7. notify new Message
	8. notify new Reply

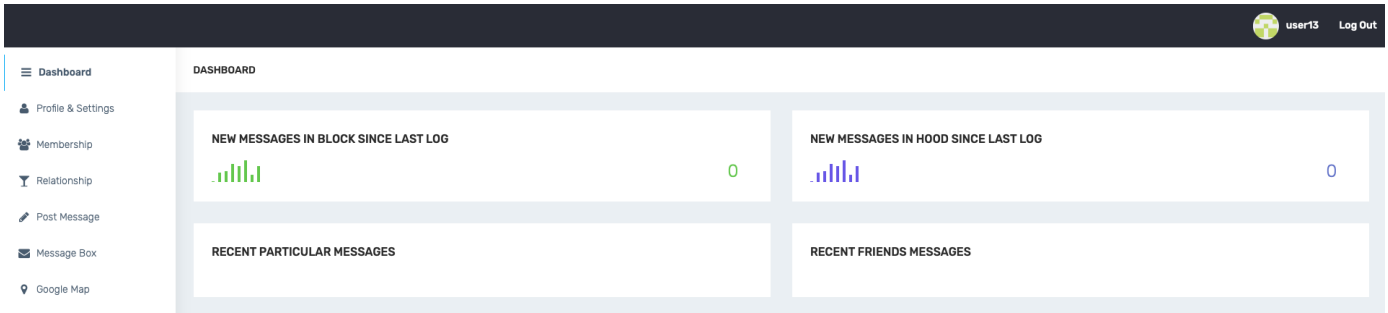
ii. Log In

This part mainly describes how users log in our system.

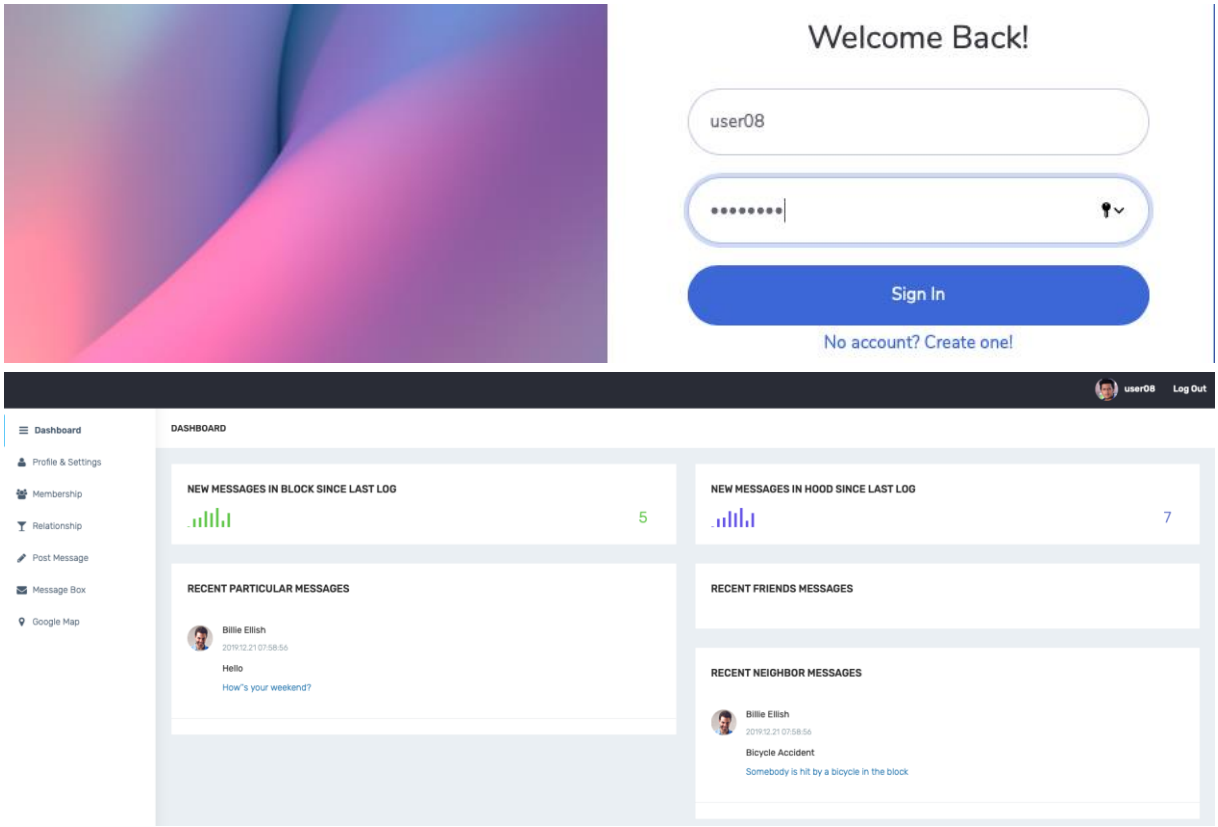
User create an account:



On successfully signed up, user is automatically logged in and redirected to the main page:



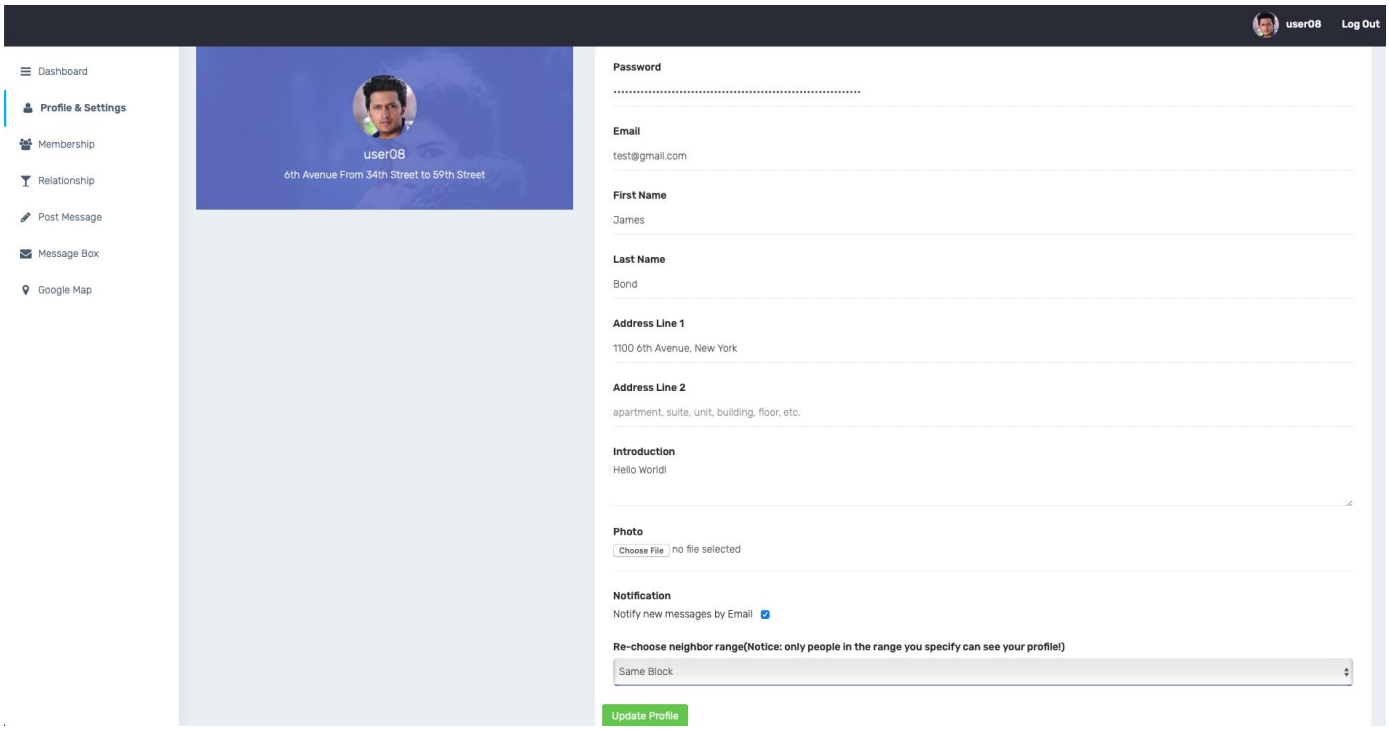
Or, users with an already existed account can log in the system directly:



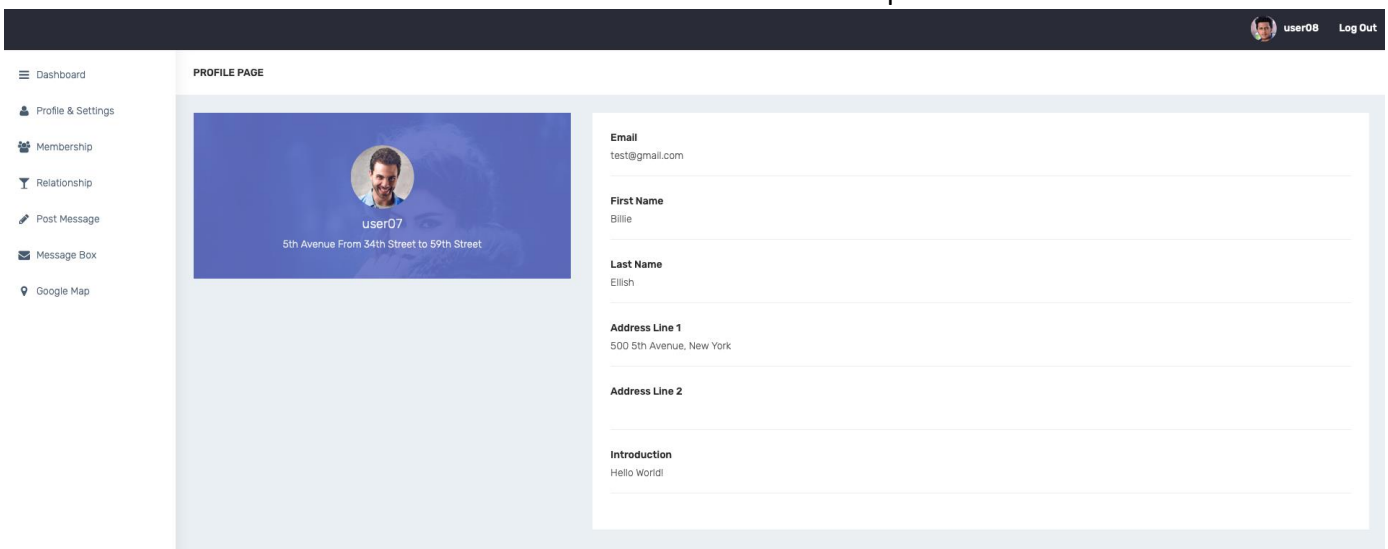
iii. Profile

This part mainly describes how users view & update their own profile, and how users view other users' profile.

User can check and update his/her own profile:



Or user check click on other user's avatar or username to view their profile:



iv. Membership

This part mainly describes how users apply to join blocks, query block information, query block & hood members and leave blocks.

Whether current user belongs to a block or not, he/she can view all available block's list and decide whether to apply to become a block's member:

BLOCKS AVAILABLE

#	Block Name	Send Application
1	Jay Street	Send
2	Gold Street	Send
3	5th Avenue From 34th Street to 59th Street	Send
4	6th Avenue From 34th Street to 59th Street	Send

If current user has already joined a block, he/she can view current block's information and members information:

CURRENT BLOCK INFORMATION

Block ID: 4
Block Name: 6th Avenue From 34th Street to 59th Street
Hood Name: Midtown Manhattan

BLOCK MEMBERS

UID	Real Name	Username
8	James Bond	@user08
9	Adam Levine	@user09
10	Bruno Mars	@user10
11	Scarlett Johansson	@user11

HOOD MEMBERS

UID	Real Name	Username
7	Billie Ellish	@user07
8	James Bond	@user08
9	Adam Levine	@user09
10	Bruno Mars	@user10
11	Scarlett Johansson	@user11

If current user has already joined a block, he/she can leave the block at any time:

LEAVE CURRENT BLOCK

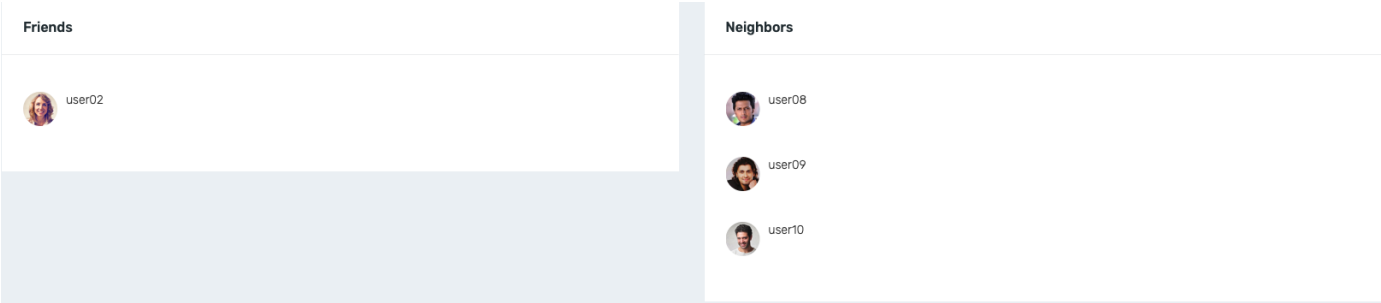
If you plan to move to another block to reside, you may leave your current block here and apply to join another block later.
WARNING: All your current neighbors will be deleted.

Leave Block

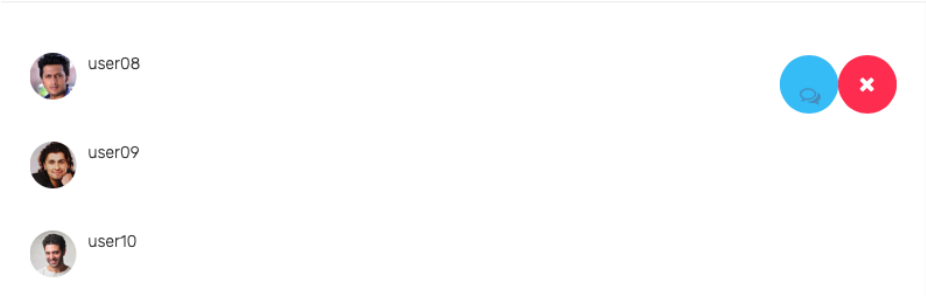
v. Relationship

This part mainly describes how users query all their friends & neighbors, send messages to them, delete them, receive new friend application and add new relationship.

User can view all his/her friends and neighbors:

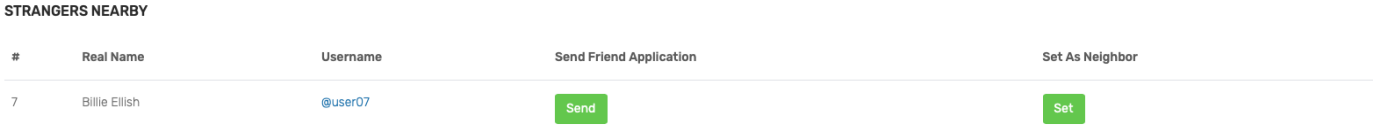


For each friend/neighbor, user can directly send a message to him/her, or delete him/her:



User may receive other user's friend application:

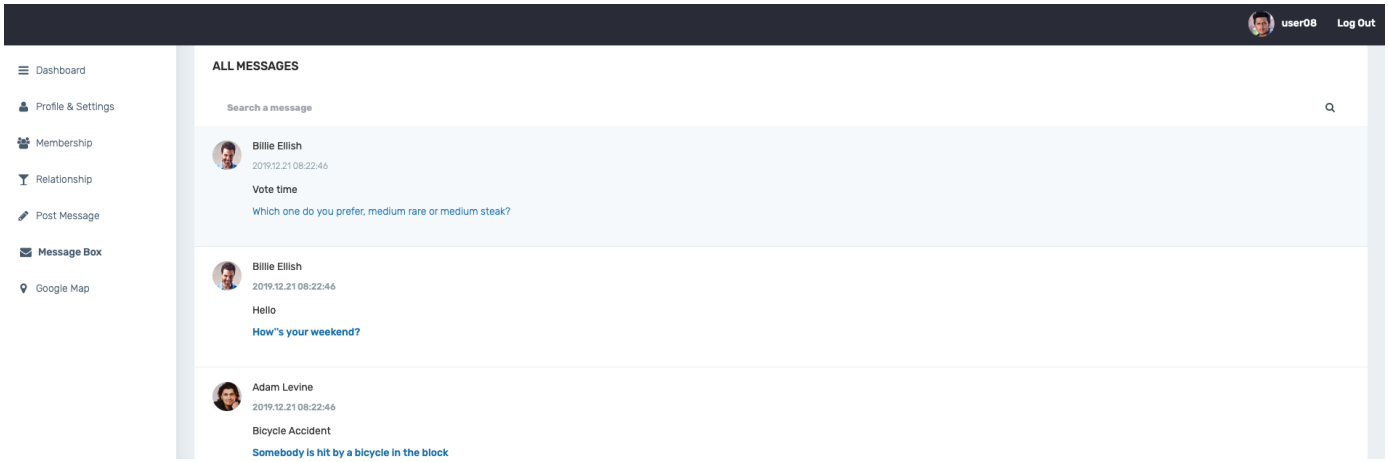
User can also query strangers in his/her hood, so as to decide to make new friends / set new neighbors:



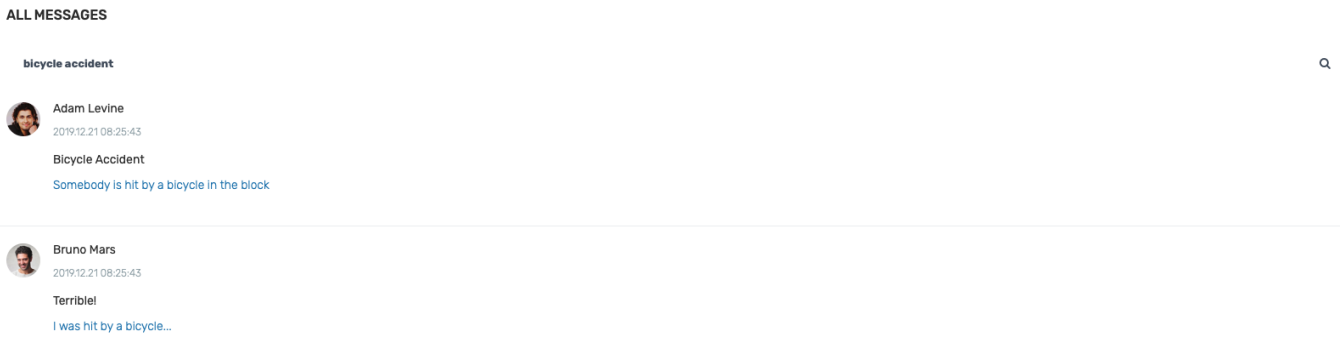
vi. Message

This part mainly describes how users query all messages in their mailboxes, post new messages, view messages in detail and reply to messages.

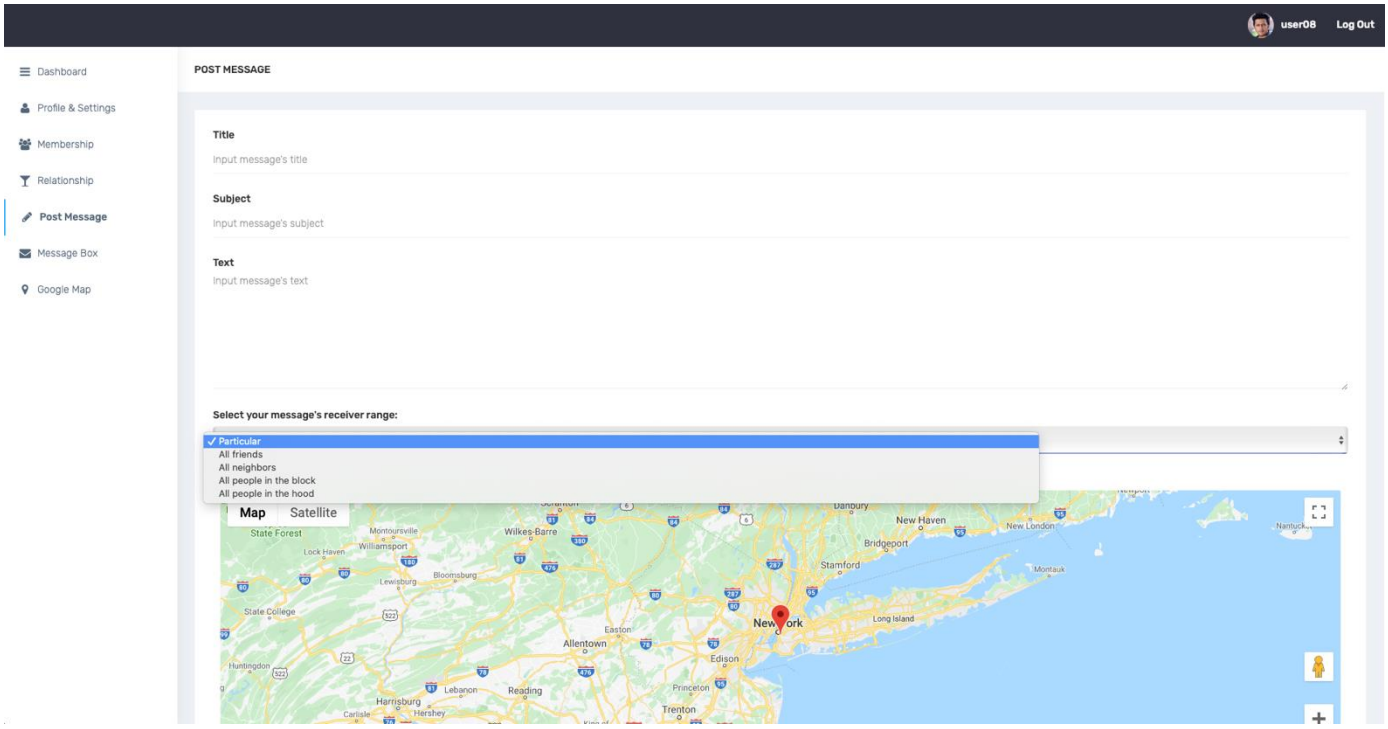
Each user has a mailbox which stores all messages the user received and posted (messages in bold font represent “unread” messages):



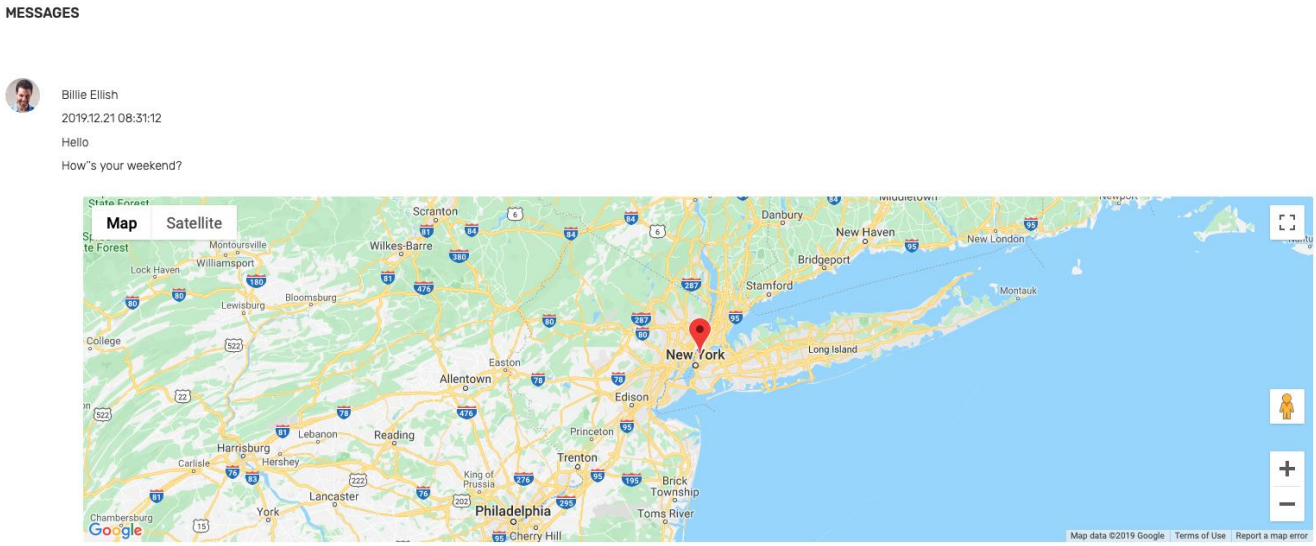
User can search certain messages using input keywords (bicycle):



User can post a new message with title, subject, text, location to certain range’s receiver:



User can view each posted message in detail:



User can reply to a message (even if he/she is the author):