# Project 2 Oingo

**Zhengxi Tian - zt586 - N11755282**

**Xinyu Xiao - xx869 - N13719972**

## Introduction

Oingo is a platform that allows users to share useful information via their mobile devices based on social, geographic, temporal, and keyword constraints. The main idea in oingo is that users can publish information in the form of short notes, and then link these notes to certain locations and certain times. Other users can then receive these notes based on their own location, the current time, and based on what type of messages they want to receive.

## Database Management System

### Relational Database

```
-- ----------------------------
-- Relational Database
-- ----------------------------
UserInfo (uid, uname, email, password, sex, city)
primary key: uid

UserMoment (uid, longtitude, latitude, state)
primary key: uid
-- ----------------------------
-- : store the current location of user, update every 5 minutes
-- ----------------------------

FriendRequest (uidFrom, uidTo, time, description)
primary key: uidFrom, uidTo, time
foreign key: uidFrom and uidTo references uid in User
-- ----------------------------
-- : time is system time when the request created, different from time in
Schedule
-- ----------------------------

Friend (fuid1, fuid2)
primary key: fuid1, fuid2
foreign key: fuid1 and fuid2 references uid in User
```

```
Note (nid, uid, sid, lid, content, postTime, tags, shareTo, cmtFlag)
primary key: nid
foreign key: uid references uid in User
             sid references sid in Schedule
             lid references lid in Location
-- ----------------------------
-- : shareTo indicates 3 cases that a note can be seen by whom:
--  `private`, `freinds`, `public`
-- : cmtFlag is bool type, which indicates whether a note can be commented
-- : tags may contain one or more tags, like `#me` or `#me#shopping#soho`
-- ----------------------------

Schedule (sid, startTime, endTime, rptFlag)
primary key: sid
-- ----------------------------
-- : rptFlag is `int` type, max value is 127 = 2^7,
--   7-bit, for repeat date or time like `every Friday`: 0000100
-- : if the schedule is not a range, only use startTime, endTime is null
-- ----------------------------

Location (lid, lname, latitude, longtitude, radius)
primary key: lid
-- ----------------------------
-- : location is modeled as a circle, point plus radius around it
-- ----------------------------

Filter (fid, sid, lid, tag, state, uid, shareFrom)
primary key: fid
foreign key: lid references lid in Location
             sid references sid in Schedule
             uid references uid in User
-- ----------------------------
-- : uid indicates who creates the filter, a user can have multiple
filters
-- : state is varchar type, users can define their own states
-- : filter can only have one tag, it is a conjunction
-- : shareFrom indicates 3 cases that a note can be seen by whom:
--   `private`, `freinds`, `public`
-- ----------------------------

Comment (cid, nid, uid, replytocid, comment, time)
primary key: cid
foreign key: nid references nid in Note
             uid references uid in User
-- ----------------------------
-- : replytocid indicates which comment it replys to
-- : time is system time when the comment created, different from time in
Schedule
```

```
-- ----------------------------
```

## Explaination for Database

### 1. User

For user, it has 2 tables to record the required information, one is `UserInfo`, another is `UserMoment`.

```
UserInfo (uid, uname, email, password, sex, city)
primary key: uid

UserMoment (uid, longtitude, latitude, state)
primary key: uid
```

- `UserInfo` is to store the basic information of a user, in which `uid` is the primary key. A user log into the system with his or her `email`.
- `UserMoment` is to store the current location of a user, and the information will be updated every 5 minutes in back-end.

### 2. Friend

Users can send friend request to others, and after accepting by other users, their friend relaitions will be stored into the `Friend` table. All the requests are stored in `FriendRequest` table.

```
FriendRequest (uidFrom, uidTo, time, description)
primary key: uidFrom, uidTo, time
foreign key: uidFrom and uidTo references uid in User

Friend (fuid1, fuid2)
primary key: fuid1, fuid2
foreign key: fuid1 and fuid2 references uid in User
```

### 3. Note

All notes published by users are stored in the `Note` table shown as below:

```
Note (nid, uid, sid, lid, content, postTime, tags, shareTo, cmtFlag)
primary key: nid
foreign key: uid references uid in User
             sid references sid in Schedule
             lid references lid in Location
```

- `nid` is the primary key, and `uid`, `sid` and `lid` are foreign keys.
- `content` is a varchar type, maximum length is 255, aims to store the content of a note.

- `postTime` is the time when a note is published, which is different from the schedule. `sid` indicates a periods of time a note can be seen.
- `tags` is a varchar type, containing multiple tags. Each tag starts at `#`. No space between tags.
- `ShareTo` indicates 3 cases that a note can be seen by whom: `'private'`, `'friends'` and `'public'`.
- `cmtFlag` is a boolean type, which indicates whether a note can be commented by others. `Y` for true, `N` for false.

**4. Schedule**

`Schedule` table is to store the specific schedule time.

```
Schedule (sid, startTime, endTime, rptFlag)
primary key: sid
```

- `sid` is the primary key.
- `startTime` and `endTime` are all datetime type, if the schedule time is a period of time, both attributes are not null. If the schedule time is a point, then `endTime` is null, only use `startTime`.
- `rptFlag` is a 7-bit binary, the corresponding bit indicates which day should be repeated. Monday to Sunday starts from high-bit to low-bit. For example, if `rptFlag = 1010100`, then it defines that the repeat days are Mon. Wed. and Fri.

**5. Location**

Location is modeled as a circle, point plus radius around it.

```
Location (lid, lname, latitude, longtitude, radius)
primary key: lid
```

- `lid` is the primary key, `lname` is the name of a location, like 'SOHO' and 'flushing'.

**6. Filter**

Filter is used to filter notes by users.

```
Filter (fid, sid, lid, tag, state, uid, shareFrom)
primary key: fid
foreign key: lid references lid in Location
             sid references sid in Schedule
             uid references uid in User
```

- `fid` is the primary key, `lid`, `sid` adn `uid` are foriegn keys.
- `uid` indicates who created the filter, that is to say, which user wants to filter what kind of notes he or she prefers to accept.
- `state` is a varchar type, users can define their own states, like 'happy holiday'.

- A filter can only have one `tag`, it is a conjunction.
- `shareFrom` indicates 3 cases that a note published by whom: `'private'`, `'friends'` and `'public'`.

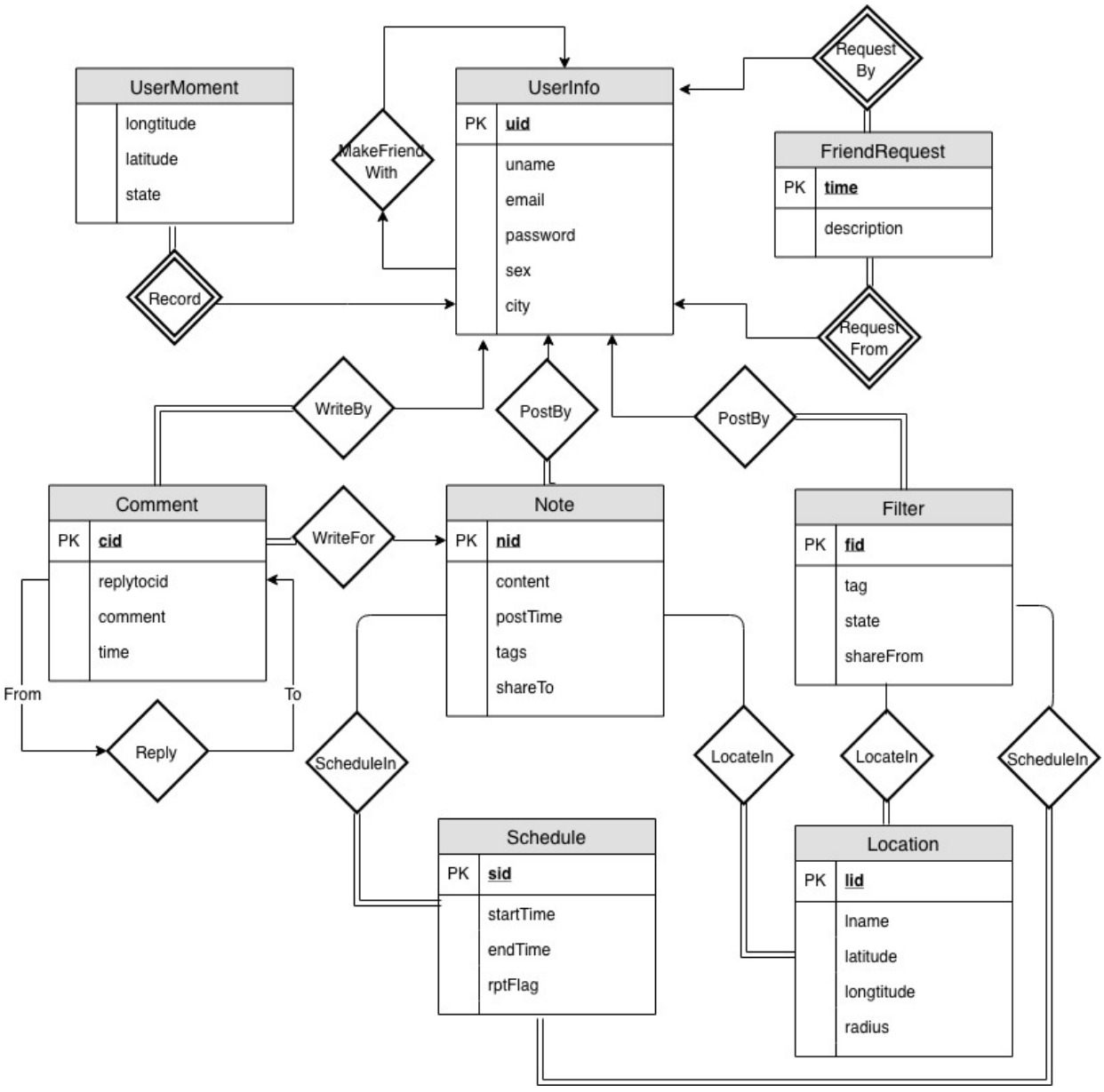
**7. Comment**

Users can comment on notes, and keep commenting on comments like twitter or facebook.

```
Comment (cid, nid, uid, replytocid, comment, time)
primary key: cid
foreign key: nid references nid in Note
             uid references uid in User
```

- `replytocid` indicates which comment it reply to.
- `time` is system time when a comment created, which is different from time in `Schedule`.

# ER Diagram




## Build Database

```
-- -----------------------------
-- BUILD DATABASE
-- -----------------------------
-- -----------------------------
-- Create Schema
-- -----------------------------
DROP DATABASE IF EXISTS Oingo;
CREATE SCHEMA Oingo;
USE Oingo;

-- -----------------------------
-- Table Structure for `UserInfo`
-- -----------------------------
DROP TABLE IF EXISTS `Userinfo`;
```

```
CREATE TABLE `UserInfo` (
    uid INT AUTO_INCREMENT PRIMARY KEY,
    uname VARCHAR(45) NOT NULL,
    email VARCHAR(45) NOT NULL,
    password INT(8) NOT NULL,
    sex VARCHAR(6) NOT NULL,
    city VARCHAR(45) NOT NULL);

-- ----------------------------
-- Table Structure for `UserMoment`
-- ----------------------------
DROP TABLE IF EXISTS `UserMoment`;
CREATE TABLE `UserMoment` (
    uid float NOT NULL,
    latitude float NOT NULL,
    longtitude float NOT NULL,
    state VARCHAR(45),
    PRIMARY KEY (uid));

-- ----------------------------
-- Table structure for `FriendRequest`
-- ----------------------------
DROP TABLE IF EXISTS `FriendRequest`;
CREATE TABLE `FriendRequest` (
  `uidFrom` INT NOT NULL,
  `uidTo` INT NOT NULL,
  `time` datetime NOT NULL,
  `description` VARCHAR(45),
  PRIMARY KEY (`uidFrom`, `uidTo`, `time`));

-- ----------------------------
-- Table structure for `Friend`
-- ----------------------------
DROP TABLE IF EXISTS `Friend`;
CREATE TABLE `Friend` (
  `fuid1` INT NOT NULL,
  `fuid2` INT NOT NULL,
  PRIMARY KEY (`fuid1`, `fuid2`));

-- ----------------------------
-- Table Structure for `Note`
-- ----------------------------
DROP TABLE IF EXISTS `Note`;
CREATE TABLE `Note` (
    nid INT AUTO_INCREMENT PRIMARY KEY,
    uid INT NOT NULL,
    sid INT NOT NULL,
    lid INT NOT NULL,
    content VARCHAR(255),
```

```
    postTime DATETIME,
    tags VARCHAR(255),
    shareTo VARCHAR(7),
    cmtFlag CHAR(1)); # Y for True, N for False

-- ----------------------------
-- Table structure for `Schedule`
-- ----------------------------
DROP TABLE IF EXISTS `Schedule`;
CREATE TABLE `Schedule` (
  `sid` INT AUTO_INCREMENT,
  `startTime` datetime NOT NULL,
  `endTime` datetime,
  `rptFlag` BINARY(7),
  PRIMARY KEY (`sid`));

-- ----------------------------
-- Table structure for `Location`
-- ----------------------------
DROP TABLE IF EXISTS `Location`;
CREATE TABLE `Location` (
  `lid` INT AUTO_INCREMENT,
  `lname` VARCHAR(45),
  `latitude` float NOT NULL,
  `longtitude` float NOT NULL,
  `radius` float,
  PRIMARY KEY (`lid`));

-- ----------------------------
-- Table Structure for `Filter`
-- ----------------------------
DROP TABLE IF EXISTS `Filter`;
CREATE TABLE `Filter` (
    fid INT AUTO_INCREMENT PRIMARY KEY,
    sid INT,
    lid INT,
    tag VARCHAR(45),
    state VARCHAR(45),
    uid INT NOT NULL,
    shareFrom VARCHAR(7));

-- ----------------------------
-- Table Structure for `Comment`
-- ----------------------------
DROP TABLE IF EXISTS `Comment`;
CREATE TABLE `Comment` (
    cid INT AUTO_INCREMENT PRIMARY KEY,
    nid INT NOT NULL,
    uid INT NOT NULL,
```

```
    replytocid INT,
    comment VARCHAR(255),
    time DATETIME);
```

## Populate Database with Sample Data

```
-- ----------------------------
-- CREATE DATA
-- ----------------------------
-- ----------------------------
-- Records of UserInfo
-- ----------------------------
INSERT INTO UserInfo (uname, email, password, sex, city) VALUES
    ('Antonio', 'antonio@nyu.edu', 11111111, 'Male', 'New York'),
    ('Mary', 'mary@nyu.edu', 22222222, 'Female', 'San Francisco'),
    ('Tom', 'tom@nyu.edu', 33333333, 'Male', 'Seattle'),
    ('Henry', 'henry@nyu.edu', 44444444, 'Male', 'Mountain View'),
    ('Kelly', 'kelly@nyu.edu', 55555555, 'Female', 'Jersey City'),
    ('Cindy', 'cindy@nyu.edu', 66666666, 'Female', 'Jersey City'),
    ('Huihui', 'huihui@nyu.edu', 77777777, 'Male', 'New York'),
    ('Kenneth', 'kenneth@nyu.edu', 88888888, 'Male', 'Washington'),
    ('Michelle', 'michelle@nyu.edu', 99999999, 'Female', 'Washington'),
    ('Casey', 'casey@nyu.edu', 00000000, 'Female', 'Jersey City');

-- ----------------------------
-- Records of UserMoment
-- ----------------------------
INSERT INTO UserMoment (uid, latitude, longtitude, state) VALUES
    (7, 40.7674987, -73.833079, 'happy holiday'),
    (8, 40.7674987, -73.833079, 'find place to eat'),
    (5, 32.844017, -97.1430671, 'happy holiday'),
    (6, 32.844017, -97.1430671, 'shopping'),
    (10, 40.6942036,-73.986579, 'enjoy weekend');

-- ----------------------------
-- Records of FriendRequest
-- ----------------------------
INSERT INTO FriendRequest (uidFrom, uidTo, time, description) VALUES
    (5, 6, "2017-11-25 19:25:35", "Im ur daddy:)"),
    (6, 10, "2017-11-25 19:25:35", "Im gakki >3<"),
    (4, 5, "2017-11-25 19:25:35", "Im stupid Henry"),
    (10, 5, "2017-11-25 19:25:35", "Im cute gakki~"),
    (7, 6, "2017-11-25 19:25:35", "Im ur big bro"),
    (5, 7, "2017-11-25 19:25:35", "Im ur big bro too"),
    (8, 9, "2017-11-25 19:25:35", "Im ur boyfriend"),
    (1, 3, "2017-11-25 19:25:35", "Hi, Im Antonio"),
    (3, 2, "2017-11-25 19:25:35", NULL),
```

```
    (1, 5, "2017-11-25 19:25:35", "Hi, Im Antonio");

-- ----------------------------
-- Records of Friend
-- ----------------------------
INSERT INTO Friend (fuid1, fuid2) VALUES
    (5, 6),
    (6, 10),
    (10, 5),
    (7, 6),
    (5, 7),
    (8, 9),
    (7, 8),
    (1, 3);

-- ----------------------------
-- Records of Note
-- ----------------------------
INSERT INTO Note (uid, sid, lid, content, postTime, tags, shareTo,
cmtFlag) VALUES
 (1, 1, 1, "It's BlackFriday! Don't miss 20% discount in Taste Collection!
", now(), "#BlackFriday#food", "public", 'Y'),
 (2, 2, 2, "Because November is #vegan month, we're celebrating
meatlessmonday with this vegan #ramen! Find this awesome ramen restaurant:
South Street Fish & Ramen Co", "2018-11-01 00:00:00", "#vegan#food",
"public", 'Y'),
 (3, 3, 3, "I love the Kiss Taco awesome charred chicken with heavenly
white cheese and a tasty corn tortilla.", now(), "#taco#food","public",
'Y'),
 (4, 4, 4, "Fiveguys helps me get through final when the time is limites.
Eating it everydayO.O", "2017-11-25 19:25:35" ,"#fiveguys#food#final",
"public", 'Y'),
 (5, 5, 5, "Cindy and I went to the Bari last night! It tastes so great! I
love dating with my babe Cindy!", now(), "#bff#koreanfood", "friends",
'Y'),
 (6, 6, 6, "Kelly and I bought lots of comestics and clothes at 34th
Street. Getting poor;( but so happy!", now(), "#BlackFriday#macys",
"public", 'Y'),
 (7, 7, 7, "Played snooker in flushing! Feel so relaxed after getting an
offer from Google.", "2018-11-21 16:00:00", "#snooker#flushing", "public",
'Y'),
 (8, 8, 7, "Does anyone in flushing now? Let's go to eat hotpot!", "2018-
11-21 16:05:00", "#me#hotpot#flushing", "friends", 'Y'),
 (9, 9, 8, "Happy Thanksgiving Night with so many friends in my home!Come
and join us!>w<", "2018-11-23 19:00:00","#Thanksgiving#frieds", "friends",
'Y'),
 (1, 10, 7, "Anyone wants to eat hotpot for lunch in Flushing?", "2018-11-
24 10:00:00", "#me#hotpot#flushing#lunch", "public", 'Y'),
```

```
 (2, 11, 9, "Welcome to NYU tandon free Thanksgiving lunch! Seats are
limited!!", "2018-11-22 09:00:00", "#Thanksgiving#lunch", "public", 'N'),
 (10, 12, 10, "Went to central park with Cindy today! Don't hesitate to
visit the park! It gets so beautiful now!", "2018-10-30 15:23:12",
"#centralpark#friends", "public", 'Y');

-- ----------------------------
-- Records of Schedule
-- ----------------------------
INSERT INTO Schedule(startTime, endTime, rptFlag) VALUES
    ("2018-11-21 00:00:00", "2018-11-26 00:00:00", NULL),      # 1
    ("2018-11-01 00:00:00", "2018-11-30 00:00:00", 1111111),   # 2
    (now(), NULL, 10101),                                      # 3
    ("2017-11-25 19:25:35", NULL, 1111111),                    # 4
    (now(), NULL, NULL),                                       # 5
    ("2018-11-21 00:00:00", "2018-11-26 00:00:00", NULL),      # 6
    ("2018-11-21 16:00:00", "2018-11-22 00:00:00", NULL),      # 7
    ("2018-11-21 16:05:00", "2018-11-22 00:00:00", 0000011),   # 8
    ("2018-11-22 00:00:00", "2018-11-23 20:00:00", NULL),      # 9
    ("2018-11-24 10:00:00", "2018-11-24 12:00:00", NULL),      # 10
    ("2018-11-22 10:00:00", "2018-11-22 12:00:00", NULL),      # 11
    ("2018-11-08 10:00:00", "2018-11-08 17:00:00", NULL),      # 12
    ("2018-11-20 00:00:00", "2018-11-26 23:59:59", NULL);      # 13 -
filter

-- ----------------------------
-- Records of Location
-- ----------------------------
INSERT INTO Location(lname, latitude, longtitude, radius) VALUES
    ("taste collection", 40.7272895, -74.04057399999999, 100),      #
Taste Collection,NY
    ("south street", 40.7272895, -74.04057399999999, 100),          #
South Street, NJ
    ("grove street", 40.7272895, -74.04057399999999, 100),          #
Tacos, NJ
    ("brooklyn", 32.844017, -97.1430671, 100),                      #
Fiveguys, BK
    ("bleeker street", 32.844017, -97.1430671, 100),                # the
Bari, NY
    ("34th street", 37.090741, -94.473863, 100),                    #
macy, NY
    ("flushing", 40.7674987, -73.833079, 1000),                     #
flushing, Queen
    ("brooklyn", 32.844017, -97.1430671, 200),                      #
Thanksgiving, BK
    ("brooklyn", 40.6942036, -73.986579, 1000),                     #
Tandon, BK
    ("central park", 40.6942036,-73.986579, 100);                   #
Central Park, NY
```

```
-- ----------------------------
-- Records of Filter
-- ----------------------------
INSERT INTO Filter (sid, lid, tag, state, uid, shareFrom) VALUES
    (NULL, 7, "#flushing", "to relax", 7, 'public'),
    (7, 7, "#hotpot", "find place to eat", 7, 'friends'),
    (9, NULL, "#Thanksgiving", "in holiday", 5, 'public'),
    (NULL, NULL, "#Thanksgiving", NULL, 6, 'public'),
    (13, NULL, "#BlackFriday", "happy holiday", 10, 'friends'),
    (13, NULL, "#Thanksgiving", "happy holiday", 10, 'public'),
    (NULL, NULL, NULL, NULL, 1, 'private'),
    (7, 7, "#flushing", "happy holiday", 7, 'friends');

-- ----------------------------
-- Records of Comment
-- ----------------------------
INSERT INTO Comment (nid, uid, replytocid, comment, time) VALUES
    (8, 7, NULL, "I'm in flushing now, how about having hotpot together
dude?", "2018-11-21 16:06:48"),
    (8, 8, 1, "Great! Where are u now?", "2018-11-21 16:07:02"),
    (8, 7, 2, "I'm at Jmart now, plz pm", "2018-11-21 16:07:34"),
    (5, 6, NULL, "love u, kiss u >3<", now()),
    (5, 5, 4, "love u too my bff!", now() + 1);
```

# Full-stack Project

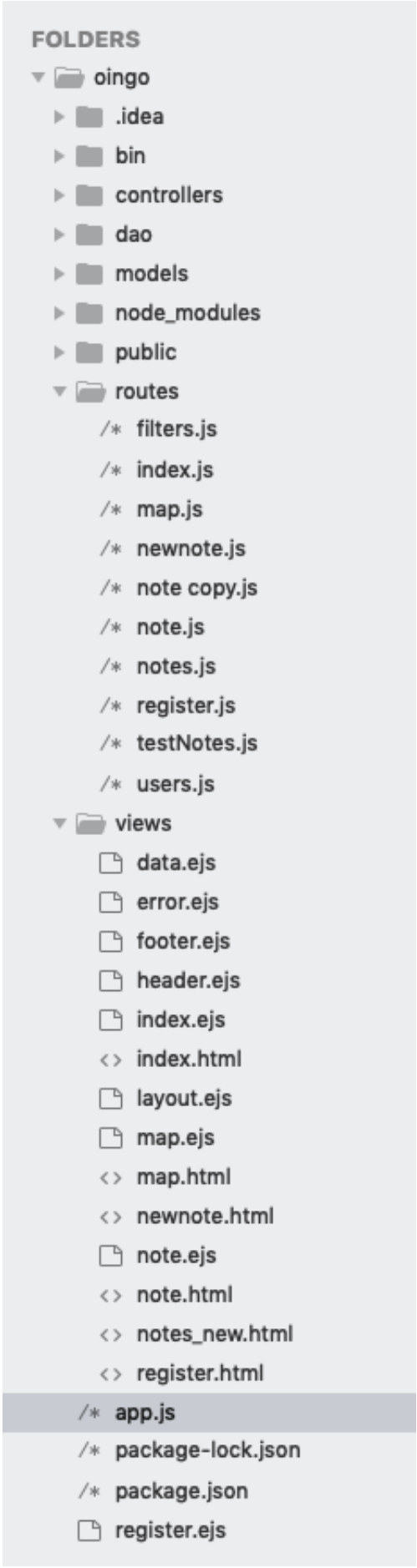
## Front-end

We use **HTML5**, **CSS**, **Bootstrap** and **EJS template** to implement front-end.

## Back-end

We use **Node.js** to implement back-end.

## Structure


FOLDERS
oingo
.idea
bin
controllers
dao
models
node_modules
public
routes
filters.js
index.js
map.js
newnote.js
note copy.js
note.js
notes.js
register.js
testNotes.js
users.js
views
data.ejs
error.ejs
footer.ejs
header.ejs
index.ejs
index.html
layout.ejs
map.ejs
map.html
newnote.html
note.ejs
note.html
notes_new.html
register.html
app.js
package-lock.json
package.json
register.ejs

## Interaction with Database

We build a new file named `Dao` to store all functions related to MySQL database.

As for the connection part, we write in `mysqlConf.js` as shown below.

```
module.exports = {
    mysql: {
        host: '127.0.0.1',
        user: 'root',
        password: 'mercury',
        database:'Oingo',
        //connectionLimit: 10
    }
};
```

## Functions

**1. Register**

If a user doesn't have an Oingo account before, the first thing he or she should do is to register one:) A user is asked to offer `username`, `email` and `password` to sign up. As soon as the user clicks the `SIGN UP` button, our front-edn will send encrypted password to the server, and then inserting the data into database.

**2. Log in**

If a user has an account, he can use his `email` as account name and input his `password` to log in to the Oingo. Once the above two attributes is sent to the server, our back-end will check if the `password` is correct, or the `email` is already in the database. Otherwise, it will reports the password isn't correct or the account doesn't exist.

**3. Send Friend Requests and Add Friends**

A user can send friend request to other users. By searching the `username`, the user can find and send friend requests to others. Once the request is sent, our back-end will get the data, and insert a request tuple into database (Table `FriendRequest`).

For example, user A sent a friend request to user B, B will see the request by refreshing the website. User B can choose to add friend with A or not. If B added A, the server will insert a friend relation into `Friend` table and delete the corresponding request in `FriendRequest` table.

**4. Post New Notes**

User can post notes on Oingo. When posting a new note, he is aksed to provide a paragraph of text as note `content`, the note can be share to whom: `private`, `friends` or `public`, the `radius` of the note can be seen on map, `tags` of the note, and a period of `time` when the note can be seen.

Users don't need to provide his current location, we can get his current longtitute and latitute automatically on the platform.

**5. Comments**

Uses can comment on others' notes only when the note is allowed to be commented. Otherwise, it doesn't have comment button on html page. Once the comments are created, front-end will send the data to back-end, and we will insert them to database.

**6. Add Filters**

Users can add filters to obtain the specific notes they prefer to receive. By filtering notes, users can choose `time`, `location`, `tag`, `state`, from whom: `public`, `friends`, and `private`. Notice that not all attributes are required, they can be null if users don't have specific choice. If users set all attributes in filter to null, he will receive all public notes.

## Refining Design of Previous Project

In project 1, we store the longtitude and latitude of a user's current location in database. However, in project 2, we don't store the coordinate into database anymore.

We implemented a `listener` to get the current location of users, and send the coordinate as a list `[longtitude, latitude]` to back-end and store them in a `session`.

By revising the design, it will save more space in database and be more effcient.

## REST URI

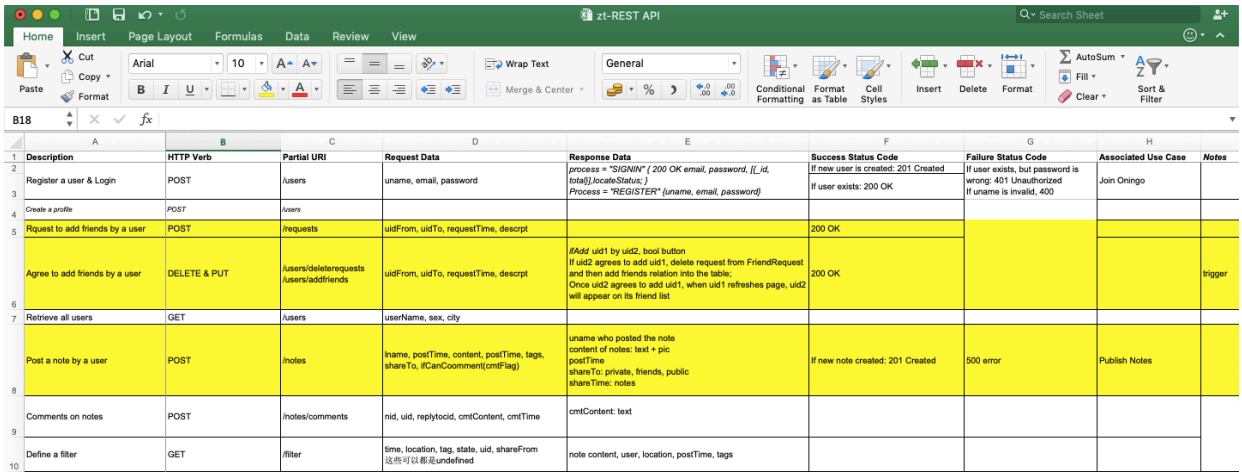
We use REST to define our URI, which is an original architectural style of the WWW.

The reason we use REST is it will minimize the coupling between client and server components via stateless operation.

The advantages to use REST can be generalized as: High performance, scalability, reliability and simplicity.

We implement it with HTTP.

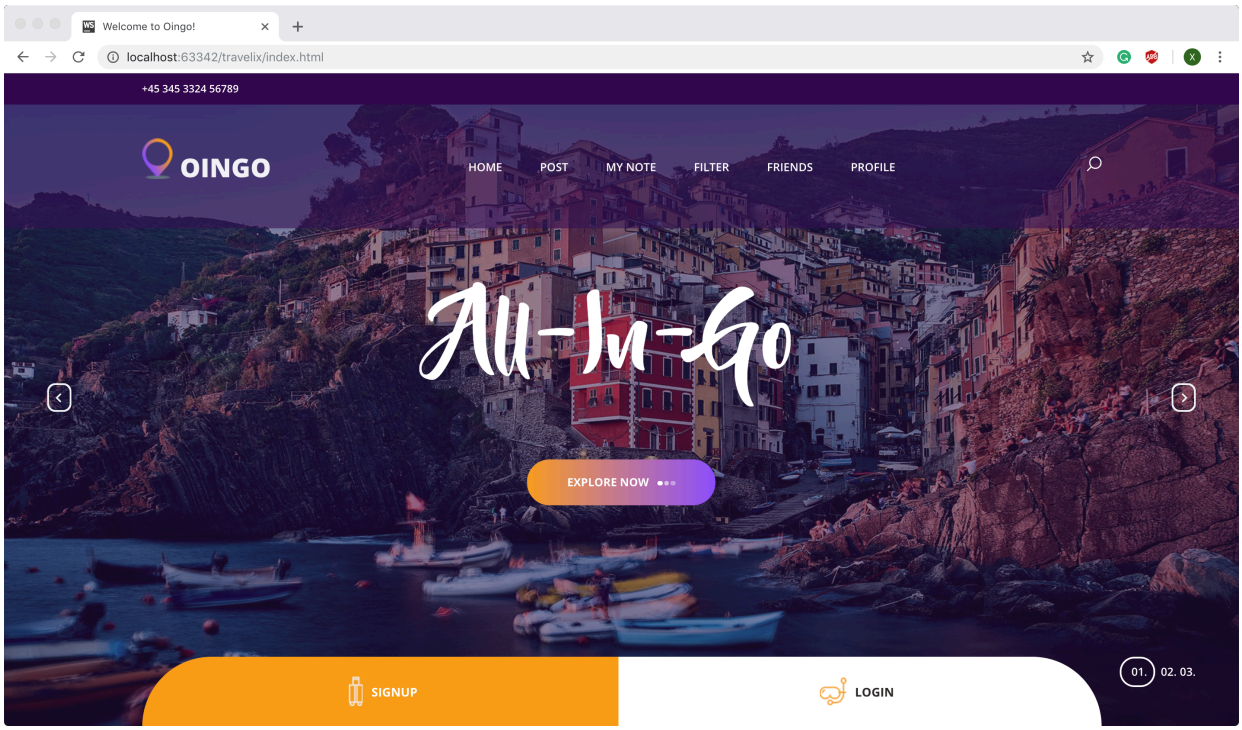
The picture shown below is a part of URIs we define in the project.



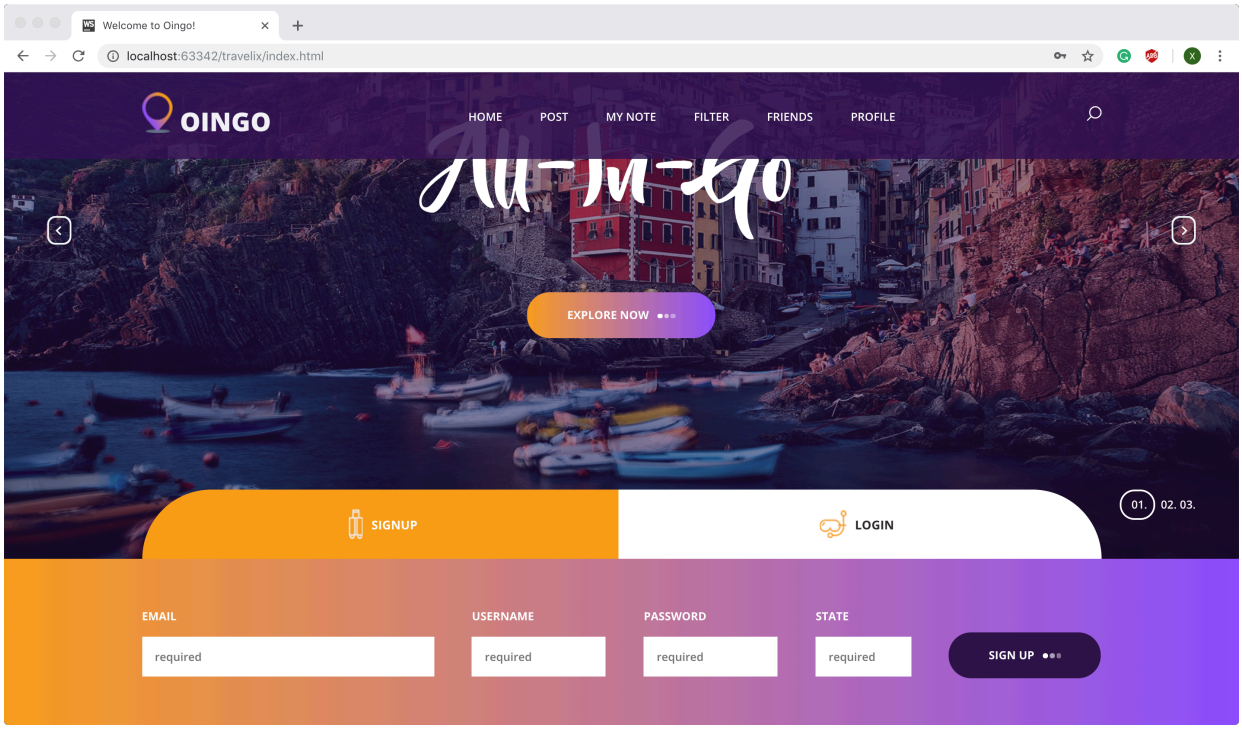
zt-REST API

| | Description | HTTP Verb | Partial URI | Request Data | Response Data | Success Status Code | Failure Status Code | Associated Use Case | Notes |
|---|---|---|---|---|---|---|---|---|---|
| 2 | Register a user & Login | POST | /users | uname, email, password | *process = "SIGNIN" { 200 OK email, password, [{_id, total}],locateStatus; }*<br>*Process = "REGISTER" {uname, email, password}* | If new user is created: 201 Created<br>If user exists: 200 OK | If user exists, but password is wrong: 401 Unauthorized<br>If uname is invalid, 400 | Join Oningo | |
| 4 | *Create a profile* | *POST* | */users* | | | | | | |
| 5 | Rquest to add friends by a user | POST | /requests | uidFrom, uidTo, requestTime, descrpt | | 200 OK | | | |
| 6 | Agree to add friends by a user | DELETE & PUT | /users/deleterequests<br>/users/addfriends | uidFrom, uidTo, requestTime, descrpt | *IfAdd* uid1 by uid2, bool button<br>If uid2 agrees to add uid1, delete request from FriendRequest and then add friends relation into the table;<br>Once uid2 agrees to add uid1, when uid1 refreshes page, uid2 will appear on its friend list | 200 OK | | | trigger |
| 7 | Retrieve all users | GET | /users | userName, sex, city | | | | | |
| 8 | Post a note by a user | POST | /notes | lname, postTime, content, postTime, tags, shareTo, ifCanCoomment(cmtFlag) | uname who posted the note<br>content of notes: text + pic<br>postTime<br>shareTo: private, friends, public<br>shareTime: notes | If new note created: 201 Created | 500 error | Publish Notes | |
| 9 | Comments on notes | POST | /notes/comments | nid, uid, replytocid, cmtContent, cmtTime | cmtContent: text | | | | |
| 10 | Define a filter | GET | /filter | time, location, tag, state, uid, shareFrom<br>这些可以都是undefined | note content, user, location, postTime, tags | | | | |

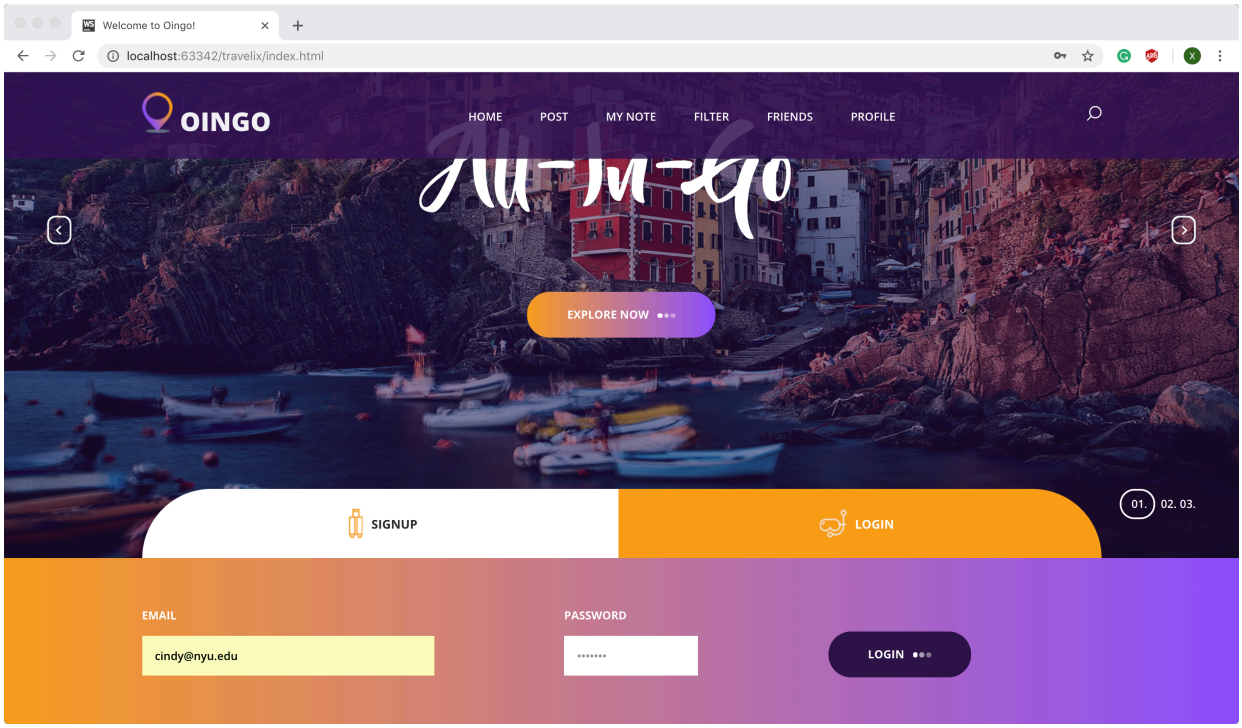
# UI Demo



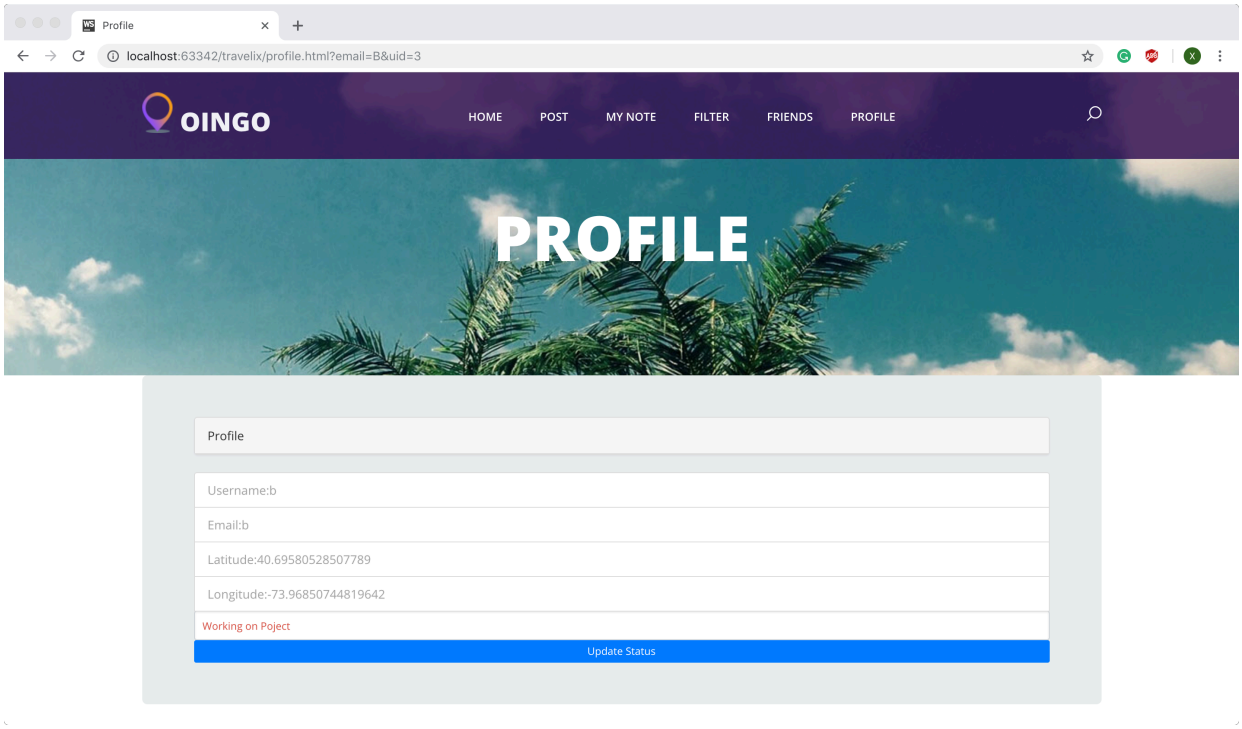

## 1. Register



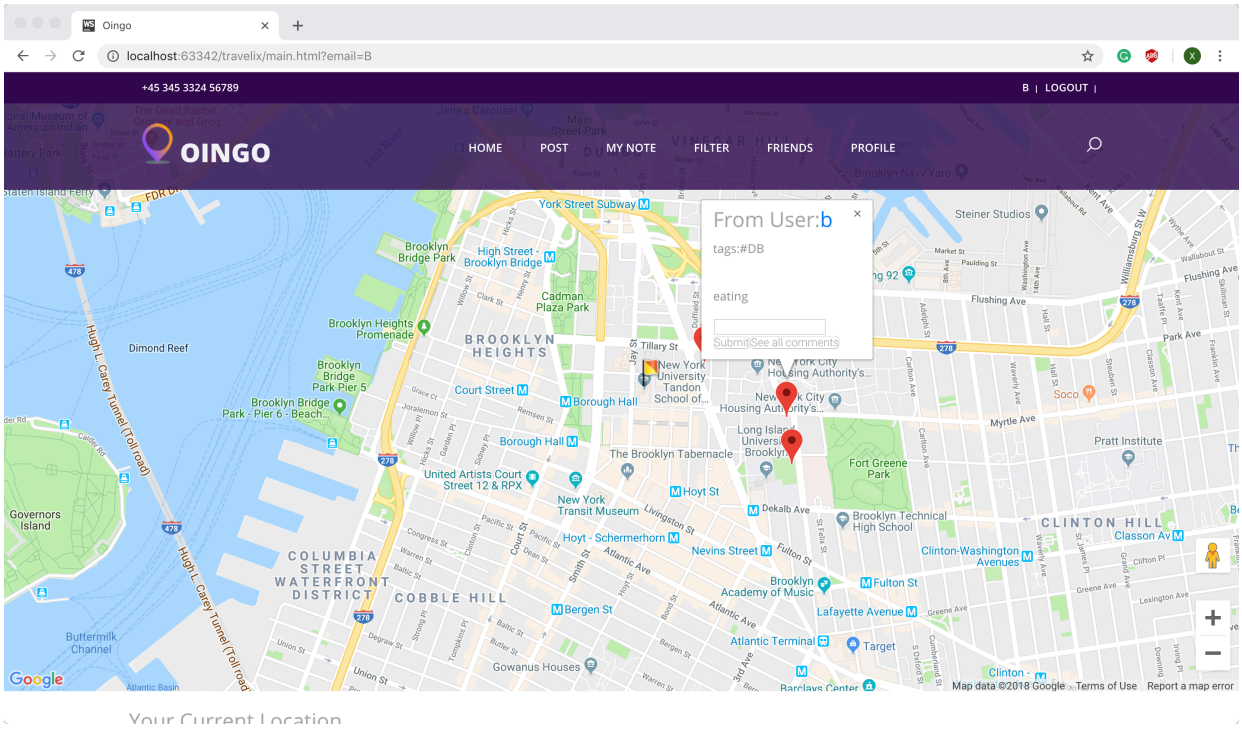

## 2. Log in



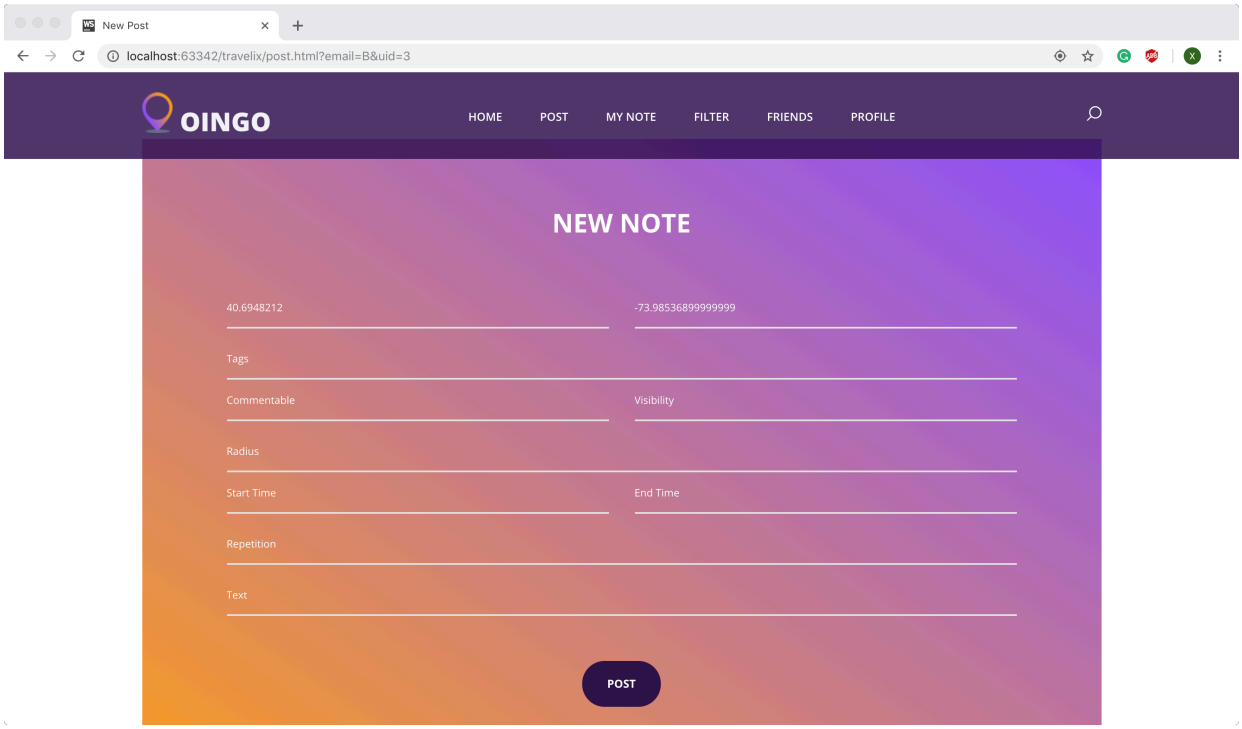

## 3. Profile



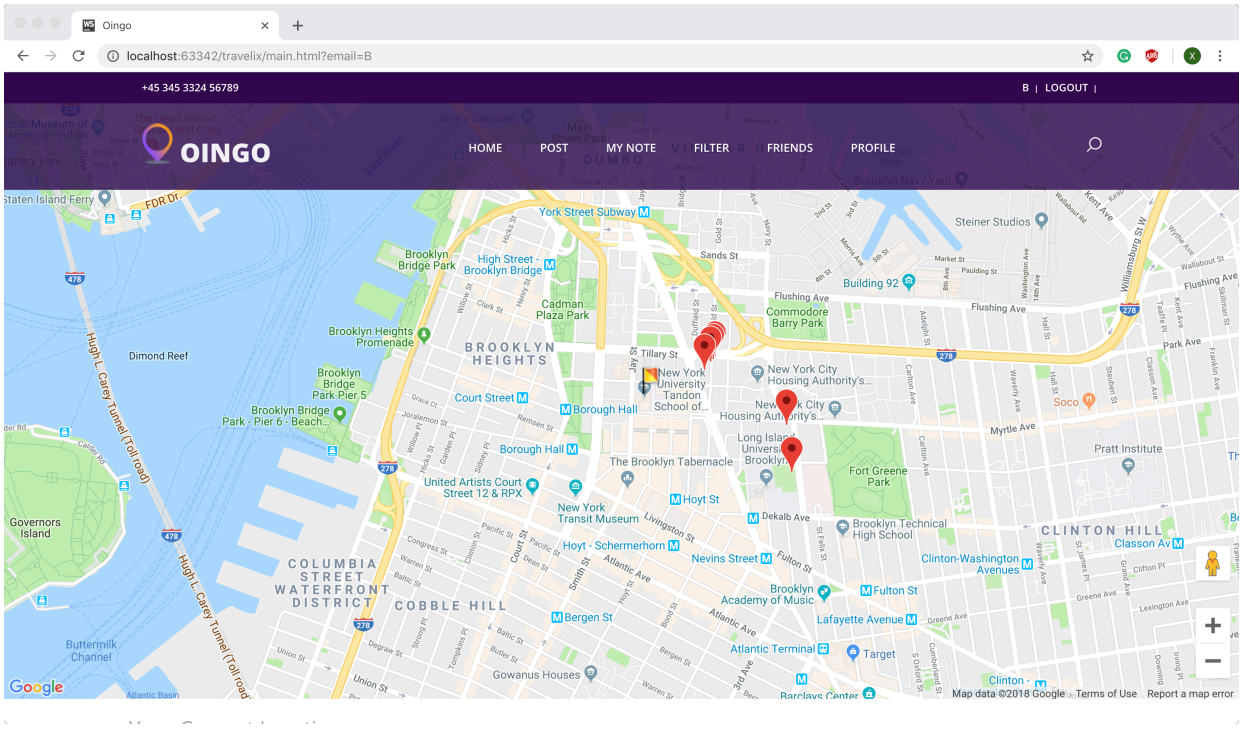

## 4. Homepage (Notes are shown on map)



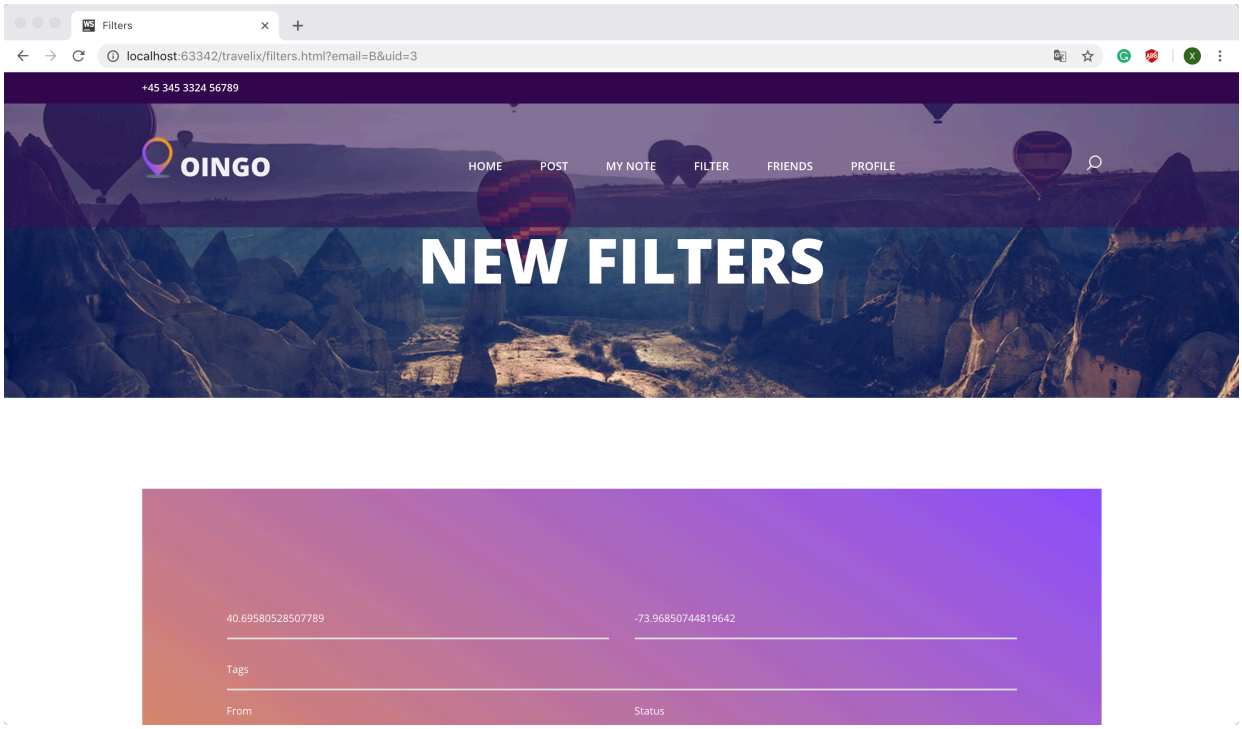

## 5. Post new notes




## 6. Click on map to obtain current location and change location



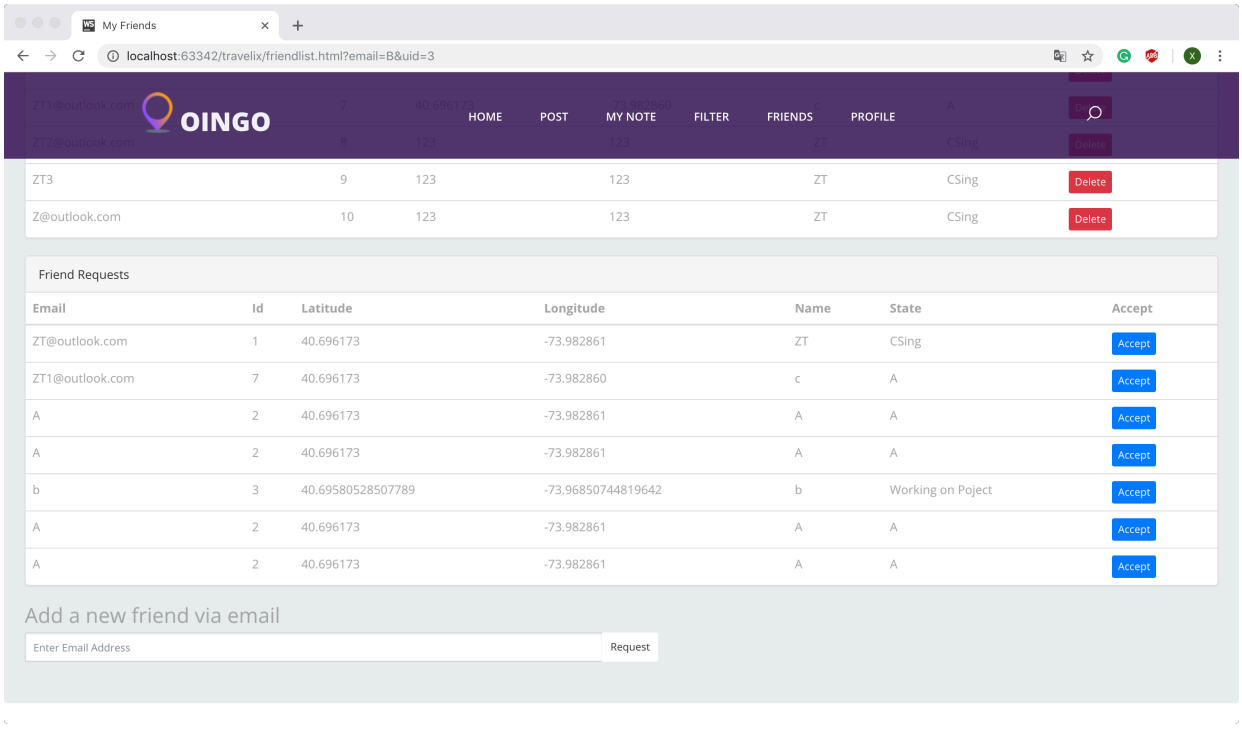

## 7. Add filters



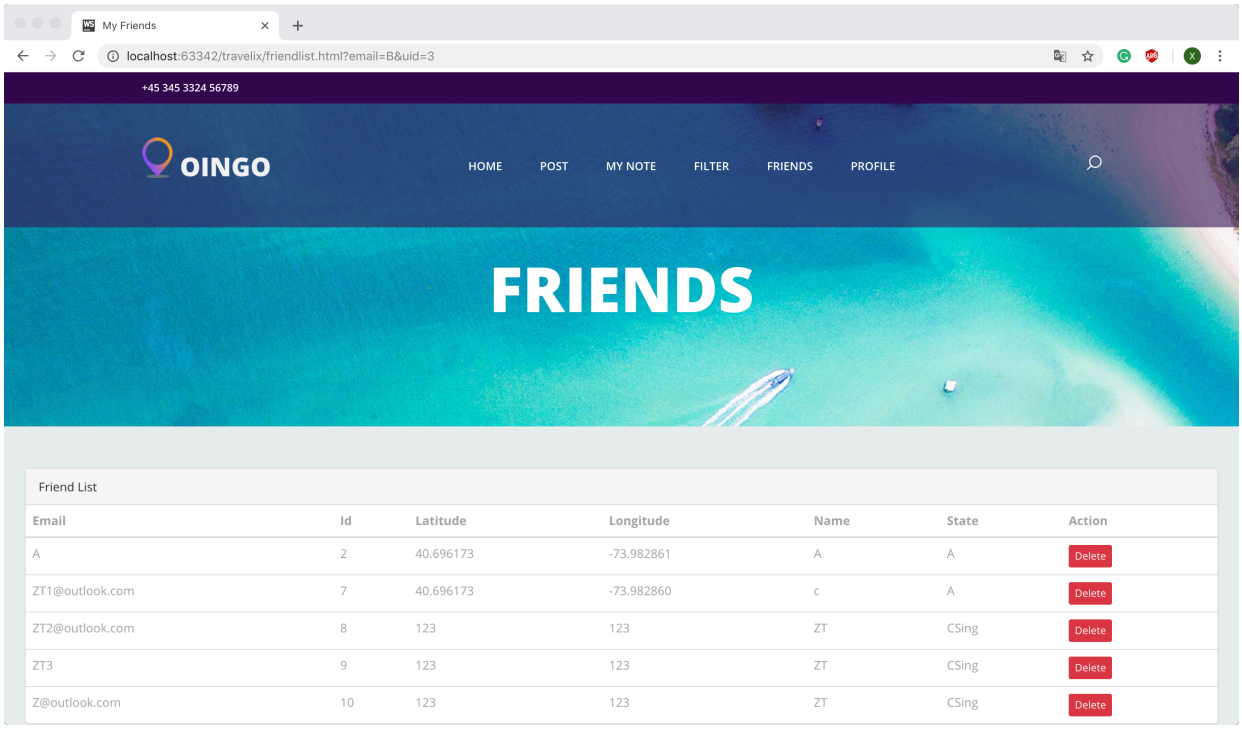




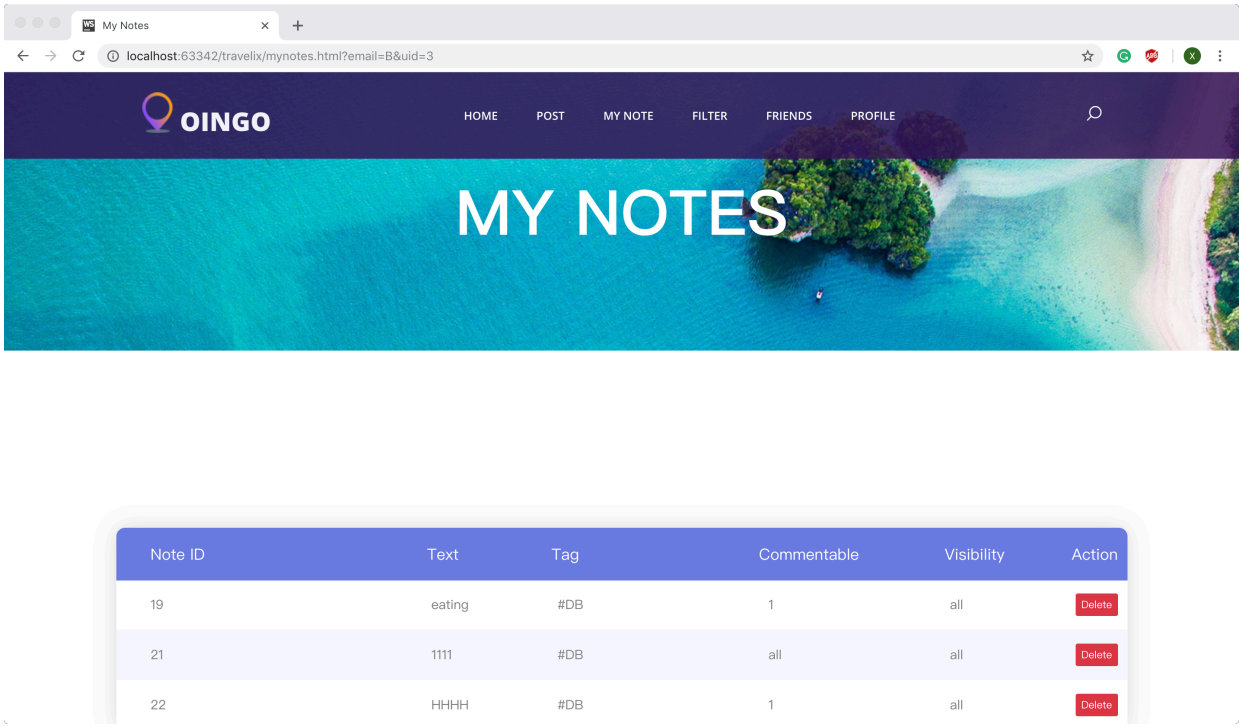
## 8. Friend requests




## 9. Friend relationships




### 10. List my notes




## Google API

We use Google Maps JavaScript API to customize maps by showing notes on it. The specific code parts are in `maps.ejs` as shown below.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<div style="width:100%; height: 445px;" id="map";></div>
<div>
    <form style="padding-top: 10px;" method="POST" id = "getMarker">
        Location_x<br>
        <input type="text" id="positionX">
        <br>
        Location_y<br>
        <input type="text" id="positionY">
        <br>
        label<br>
        <input type="text" id="type1">
        <br>

        <br>
        <button type="submit">Submit</button>
    </form>
</div>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>

    var map;
    var markersArray = [];
    var settle_marker;
    var image =
'https://developers.google.com/maps/documentation/javascript/examples/full
/images/beachflag.png';

    function initMap() {
        var uluru = {lat: 40.7675, lng: -73.8331};
        map = new google.maps.Map(document.getElementById('map'), {
            zoom: 12,
            center: uluru
        });

        // add flag, click on map, get current location
        map.addListener('click', function(e) {
            console.log(e.latLng.lat().toFixed(6),
e.latLng.lng().toFixed(6))
            // send pos to back-end
            let pos = {
                "lat" : parseFloat(e.latLng.lat().toFixed(6)),
                "lng" : parseFloat(e.latLng.lng().toFixed(6))
```

```
            }
            if (settle_marker !== undefined){
                settle_marker.setMap(null)
            }
            settle_marker = new google.maps.Marker({
                position: pos,
                map: map,
                clickable : false,
                icon : image
            });
        });
    }

    function addMarker(title, content, position) {
        var contentString = '<div id="content">'+
            '<div id="siteNotice">'+
            '</div>'+
            '<h1 id="firstHeading" class="firstHeading">' + title +
'</h1>'+
            '<div id="bodyContent">'+
            '<p>' + content + '</p>'
        '</div>'+
        '</div>';
        var infowindow = new google.maps.InfoWindow({
            content: contentString,
            maxWidth: 200
        });

        var marker = new google.maps.Marker({
            position: position,
            map: map,
            title: title
        });
        marker.addListener('click', function() {
            infowindow.open(map, marker);
        });
        markersArray.push(marker)
    }

    function removeAllMaker() {
        for (i in markersArray) {
            markersArray[i].setMap(null);
        }
        markersArray.length = 0;
    }

    function renderMarker(notes) {
        //notes: [obj]
        for (i in notes) {
```

```
            title = notes[i].title;
            content = notes[i].content;
            position = notes[i].position;
            addMarker(title, content, position);
        }
    }

    $("#getMarker").submit((content) => {
        content.preventDefault();
    let positionX = $("#positionX").val()
    let positionY = $("#positionY").val()
    let type1 = $("#type1").val()
    query = "testNotes/?positionX=" + positionX +  "&positionY=" +
positionY + "&type1=" + type1;
    console.log(query)
    $.get(query, function(result){
        // console.log(result)
        removeAllMaker()
        renderMarker(result)
    });
    })
</script>
<script async defer
        src="https://maps.googleapis.com/maps/api/js?key=AIzaSyAo-
JtxBf6i8zFh2EVERtO8gOUOEtyONy8&callback=initMap">
</script>

<% locals.posts.forEach(function(post, index){ %>
<div class="item">
    <h2>
        <a href="/single/<%= post.tags %>">
            <%= post.tags %>
        </a>
    </h2>
    <time>
        <%= post.postTime %> by:
        <a href="/author/<%= post.uname %>">
            <%= post.uname %>
        </a>
    </time>
    <p><%- post.content %></p>
    <hr/>
</div>
<% }) %>

</body>
</html>
```