Final Report

SNU MO-IM
Team 9
12/21/2017

Jaeyeon Kim
Jiyun Jeong
Pimon Atipatyakul
Seungchan Kim

1. Abstract

SNU students tend to participate in school clubs more lately. Still, there is no web service where the students can easily access to the activities and meeting. The SNULife website could be one of the choices. However, seeing that SNULife is not yet easily in well-ordered and it is still hard to categorize all the events and MO-IMs, our team had decided to create a web service 'SNU MO-IM' for only SNU students with authentication method using SNUmail and recommendation service via Facebook API.

SNU MO-IM aims to help SNU students to find other students with common interests and arrange some MO-IMs together. By using frontend and backend services, our web service now can make sure that users will have a nicer experience creating, joining, recommending the meeting to friends and even have an opportunity to make new friends to broaden up their social perspective. With SNU MO-IM, we provide a Matching Of Identical Motives.

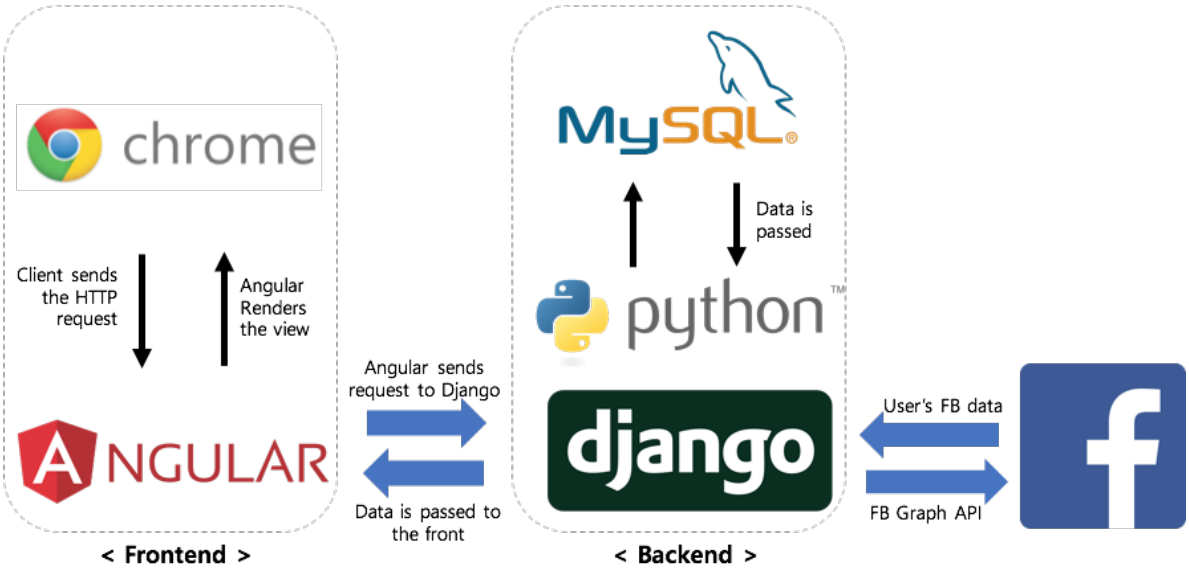
2. Introduction

SNU MO-IM categorizes MO-IMs (meetings) into several categories (i.e., Study, a student club, performance) so that users can easily find the MO-IMs they want. The two primary features of this project are creating a new meeting & participating in an active meeting (which is already created by other users). Also, this service can recommend some MO-IMs that the user might like based on interests of a user, meeting join history and the joining status of user's friends. It can also connect to Facebook to get information of the user, such as Friend status, and for the convenience of signing in. We also provide some additional features such as searching & tagging MO-IMs, and sending messages or inviting to MO-IMs between users.

3. Design

1. Design flow

- Frontend : Angular2 + Typescript
- Backend : Django + Python
- Database : MySQL
- Testing : Jasmine(Frontend) + Django testing(backend)
- Facebook API



Full Screenshots for User Interface (and Features) are included at the last pages.

2. Features
    a. User
        i. Sign Up (with mySNU authentication)
        ii. Sign In / Sign Out
        iii. View / Edit Profile
        iv. Connect to / Sign In with Facebook
    b. Facebook
        i. Authentication
        ii. Bring Profile Information
        iii. Getting Friends
    c. Meeting
        i. Create / Edit
        ii. Join / Leave / Close
        iii. Comment (Private / Public)
        iv. Searching (by title, author, category, tag, filter closed/full MO-IMs)
        v. HashTag
    d. Recommendation
        i. Recommend users, MO-IMs
    e. Message

4. Implementation

- **User**

(1) Basics

- We used Django User Auth Field for the basic model field to use authentication feature for sign in and sign out functions. However, as we needed more fields to save student information, we created another model called Ex_User (Extended User) and implemented a one-to-one relation with Django user. We saved name, college, interest subjects and friends (from Facebook) and access token for Facebook.

(2) mySNU authentication

- As our service aims for the use of Seoul National University (SNU) students, the user has to get authentication mail through mySNU mail account (@snu.ac.kr). To use this, the user has to sign in with their mySNU username. When the users complete signing in, we send an activation email to their account. The mail includes account (for our service) activation token link. When the user clicks the link, the activation process is completed, then the user can use our service.

- **MO-IM**

- A User can create a MO-IM in certain category(subject). When creating a MO-IM, a user can select maximum numbers of people who can participate. When there are empty seats in the MO-IM, other users can join the MO-IM. A member of MO-IM can also invite other users to join the MO-IM. After joining the MO-IM, the user can also leave the MO-IM. The author of the MO-IM can close or edit the detail of the MO-IM, regardless of whether the MO-IM is full or not. A user can leave a comment (public / private option is given) to each MO-IM.

- **Connection to Facebook**

(1) Basics

- We used Facebook Graph API to connect our user system with Facebook and get information. To keep using Facebook features, we had to get and update access token to Facebook of each user. Hence, we made Access token Field to save and update it. Therefore, if the user is connected to Facebook and the access token of the user is not expired, the user can sign in with Facebook itself, not having to type in our username and password.

(2) Connect our User system with Facebook

- The basic authentication system for facebook automatically allows signing in when the Facebook user doesn't have an account for our system. However, as we need mySNU authentication for signing in, we had to prevent signing in from Facebook. So we changed social authentication pipeline by deleting the sign-up process and, instead, adding a check if the user actually has an account for our service.

```
# Facebook Login
SOCIAL_AUTH_PIPELINE = (
    'social_core.pipeline.social_auth.social_details',
    'social_core.pipeline.social_auth.social_uid',
    'social_core.pipeline.social_auth.auth_allowed',
    'social_core.pipeline.social_auth.social_user',
    'snumeeting.social.check_user',
    'social_core.pipeline.social_auth.associate_user',
    'social_core.pipeline.social_auth.load_extra_data',
    'snumeeting.social.save_access_token',
    'snumeeting.social.save_friends_data',
)
```

Basically 'Connect to Facebook' button only exists in Profile page of the user, after successfully signing in. After connecting to Facebook, the access token and friend information is updated whenever the user signs in.

(3) Getting friend (and access token) Information

- Getting information from Facebook was done by Facebook Graph API. We set up our application on Facebook, so users obtain their own Facebook uid with an account with our service when they connect both accounts. With the uid and access token,

we could get much data. Still, getting information is impossible when the access token expires, so we updated the access token whenever the user signs in. The primary data we used was Friend data of the user, which we used for recommend MO-IMs to the user or to see mutual friends. We also got Facebook profile picture and name to show them in User Profile.

- **Recommendation**

(1) Basics

- We used similarity algorithm to implement our recommendation system. For computing similarity, we used user's participation history for MO-IMs in each subject. Therefore, we add 'joinHistory' field in User and College model to count the number of MO-IMs in each subject that user (or users belong to certain college) has participated. For example, if a user has participated 1 MO-IMs in english(id=2) subject and 3 MO-IMs in dance(id=4) category, the user's joinHistory will be {"2: 1, "4": 3}. A College's joinHistory would be sum of joinHistories of users belong to that College. When we compute the similarity, first we converted this json format into vector. For example, in above case, if there are total 6 subjects in the database, the converted result will be [0,1,0,3,0,0]. And we compute the similarity between vectors using **euclidean distance algorithm**.
    - if **p** = (p_1, p_2,..., p_n) and **q** = (q_1, q_2,..., q_n) are two points in Euclidean *n*-space, then the distance (d) from **p** to **q**, or from **q** to **p** is given by

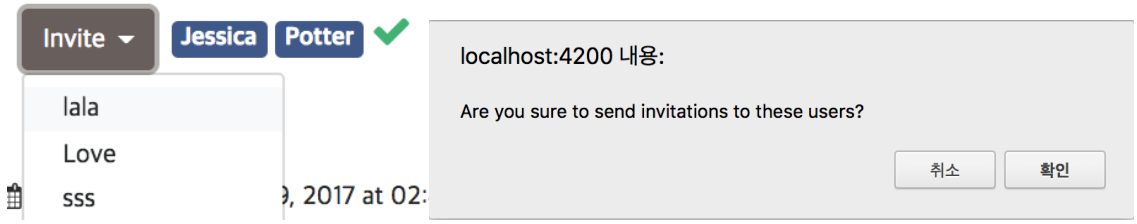
$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

After getting the distance, we computed the similarity by $1/(1+d(p,q))$

(2) Recommend Users

- To recommend users who are similar to the logined user, we not only used euclidean distance between users(d_{user}) but also used distance between the logined user's college and target user's college($d_{college}$). We put $d_{college}$to more likely recommend people in the same college among people with the same d_{user}, and to avoid cold start. After computing both distances, we compute the final distance by $d_{user} + d_{college}/3$ (We gave more weight to d_{user}). We used this result to recommend users who might also like to join a MO-IM that a user is in. So a user can choose other users to send an invitation message, and actually send a invitation message by our message system (stated below).



<invitation message>

```
def getUserSimilarity(target, K):
  # Find the user having the most similar join history
  # target : target user to calculte similarity
  # K : number of similar users to be recommended
  # return: List of tuples - [ top K (Ex_User's id, similarity score) ]

  manager = JoinHistoryManager()
  users = Ex_User.objects.all()
  similarities = {}

  jh_target = manager.convertToList(target.joinHistory)

  d_colleges= getCollegeDistance(target.college)

  for user in users:
    if user.id == target.id:
      continue
    if target.joinHistory != "{}":
      jh_user = manager.convertToList(user.joinHistory)
      jh_d = euclidean_distance(jh_target, jh_user)  # Interest Distance-
    else:  # Cold start -> computing join history similarity is meaningless
      jh_d = 0
    col_d = d_colleges[user.college_id]          # College Distance-

    # Convert distance to similarity
    # (Current weight) InterestDistance:CollegeDistance = 3:1
    similarities[user.id] = 1 / (1 + jh_d + col_d/3)--

  return heapq.nlargest(K, similarities.items(), key=itemgetter(1))-
```

<algorithm computing user similarity>

(3) Recommend MO-IMs

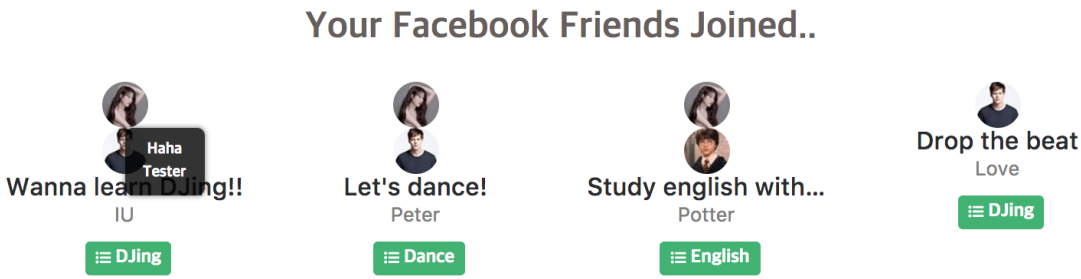
(a) MO-IMs that the user might like

- To recommend MO-IMs, we used user similarity from (2). For each MO-IMs that each similar user joined, we added up the similarity score of the similar user. After then, we gave more score to MO-IMs on the subject that the user is interested.

```
def getRecMeetings(target, N):
  # Get meetings that similar users is joined
  # N : maximum number of meetings to be recommended

  K = 3     # number of similar users to be recommended
  target_interests = target.subjects.all()
  scores = {}-

  similar_users = dict(getUserSimilarity(target,K))

  # Score based on similar users
  sim_user_ids = list(similar_users.keys())
  for uid in sim_user_ids:
    user = Ex_User.objects.get(id=uid)
    meetings_joined = list(user.meetings_joined.all())

    for m in meetings_joined:
      # exclude closed meeting or one that target user is already joined
      if not (m.is_closed or (target in m.members.all())):--
        scores[m] = similar_users[uid]

  # Score based on target user's interest
  for subject in target_interests:
    for m in subject.meetings.all():
      if not (m.is_closed or (target in m.members.all())):-
        if m in scores.keys():
          scores[m] += 0.15    # Give advantage to meetings in user's interest
        else:
          scores[m] = 0.15

  topN = heapq.nlargest(N, scores.items(), key=itemgetter(1))-

  return list(map(lambda x: convert_meeting_for_mainpage(x), [t[0] for t in topN]))-
```

<algorithm computing MO-IMs to be recommended>

(b) MO-IMs that the facebook friends

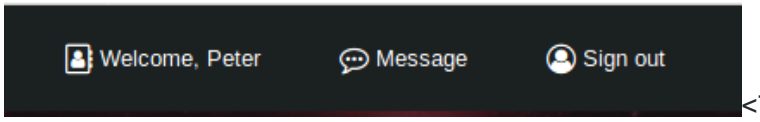


- From facebook information, we recommended MO-IMs that the logined user's facebook friends joined, and showed the profile of the friends.

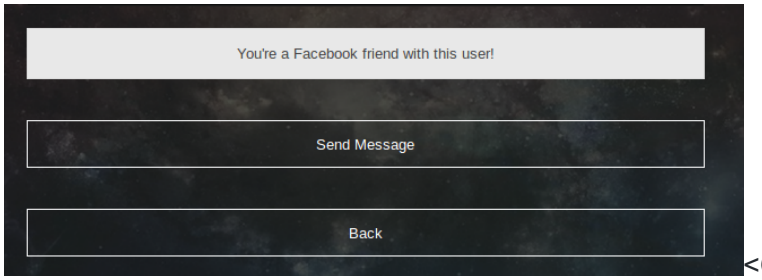
- **Private Message**

(1) Basics

- Private messages can be implemented by either chat-type feature or mail-type feature. Our private message is to communicate with each other to arrange MO-IMs well and to be more private than comments. Therefore, between the real-time chat-type feature and refresh-based mail-type feature, we implemented our private message by mail-type feature with a slight chat-type feature.
- Users can get to the message page by clicking message button on the top of all pages or in the middle of the other user's page



<Top of all pages>



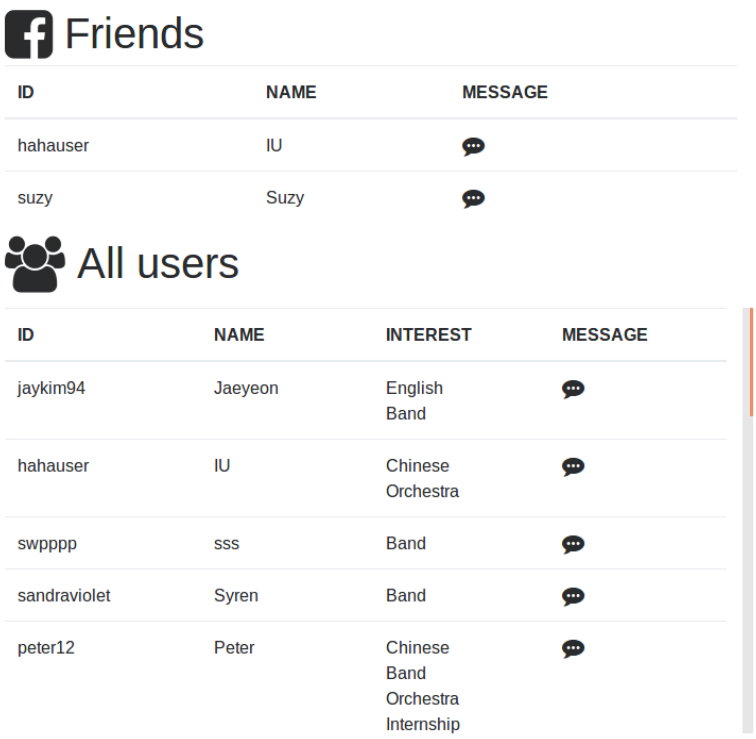
<Other user's profile page>

(2) Main page

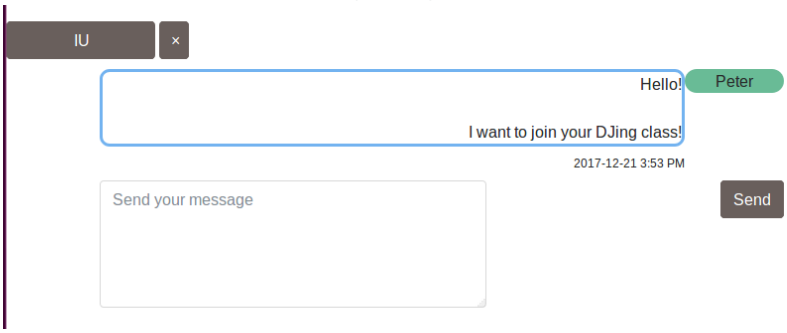
- For the main page of the message, it shows the user's Facebook friends on the top of the page. It is for arranging MO-IMs more easily with actual friends.
- For the users to arrange MO-IMs with strangers, we implemented to show all users, including the subjects each user is interested in.
- The user can see the ID, username of other users, and send message to them with the message button.
-

(3) Private message page

- Users can chat with other users privately.
- Inviting someone in the meeting detail page immediately sends the user the invitation message.
- Every user can immediately see what he/she sent.



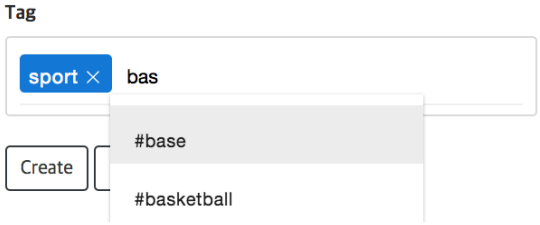
<Message page - main>



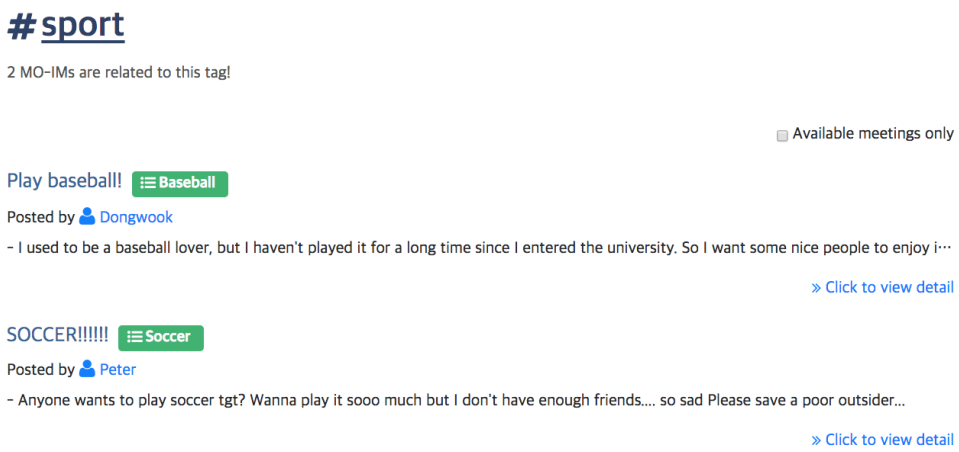
<Message page - private>

- **Hash Tag** 

- Each MO-IM can have hash tags to allow the user to apply dynamic, user-generated tagging that makes it possible for others to find MO-IMs with a specific theme or content easily.



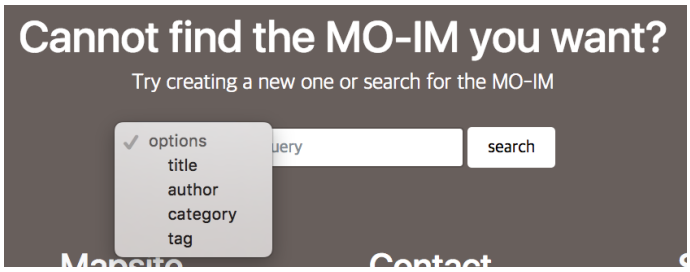
< Add/delete tags when create/edit a meeting >



<tag searching result>

- **Searching**

- We provided search engine on the main page for users to easily find MO-IMs that they want. Users can basically search by title, author, and category(subject). For category searching, the query can be left to be empty if a user wants to see every MO-IMs on a particular subject. Otherwise, the user can get a result searched both by query & filtering subject. It also provided tag searching, which shows the result same as above.



5. Lessons learned

1. The usage of Typescript and Python

- While some of us had experience of using Python, Typescript was all new for us, and we have not used them for developing actual services. Learning how to use the framework such as Angular2 and Django was very challenging but rewarding.

2. Learning how to test our work (unit testing)

- We all had to test when we develop or code, but so far we did testing manually. We didn't know testing tools, such as Karma/Jasmine or Django unit testing. With the testing tool, we could find our error more efficiently and build more robust service.

3. How to co-work with other people & How to use Github

- Until now, most us haven't had much experience working together with other people when writing codes. So at first, many of us had difficulty using Github, and the concept of things such as pull request, code review was not familiar to us. But after worked on Github with our teammates for this project, we became confident to use Github and to work with other people.

4. How web services are built & How they actually work

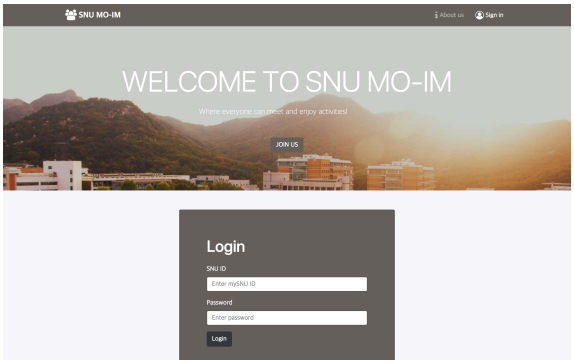
- We all knew the concept of frontend and backend, but it was really a brief knowledge, not clear. But by building a web service all by ourselves, we could know how the HTTP request/response are handled, and how the interaction between frontend and

backend is done, and how the web services are connected to other social media such as Facebook.

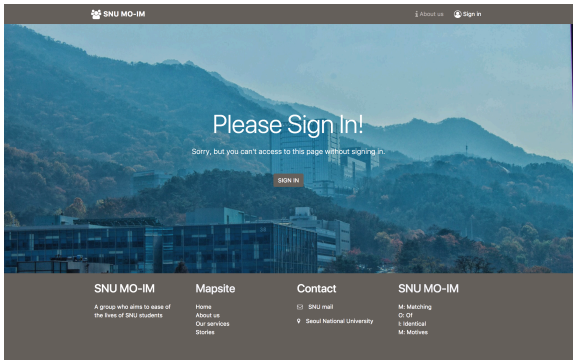
5. Apply concepts learned from other courses

- Before taking this course, some of us have taken courses such as Database, System Programming, Data Mining. However, we didn't have an actual chance to apply the things we have learned. But while building this service, we needed to apply concepts learned from Database course(foreign key, table relationship..) to design tables. And by applying similarity algorithm learned from Data Mining course, we became to know how the algorithm written in the book can actually be used. We learned 'what HTTP request/response and the web server is,' but the concept became much more clear when we built everything by ourselves.

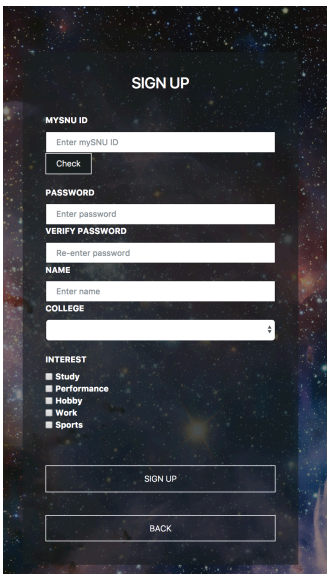
- Screenshots of our services



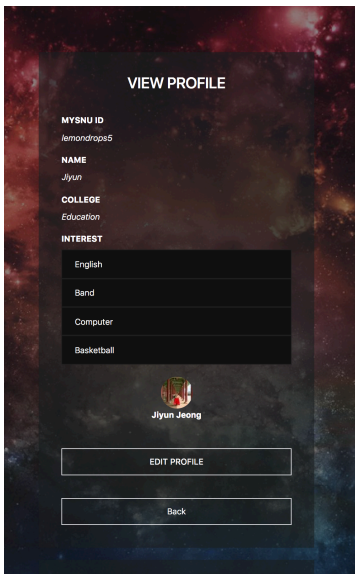
Start Page & Sign In



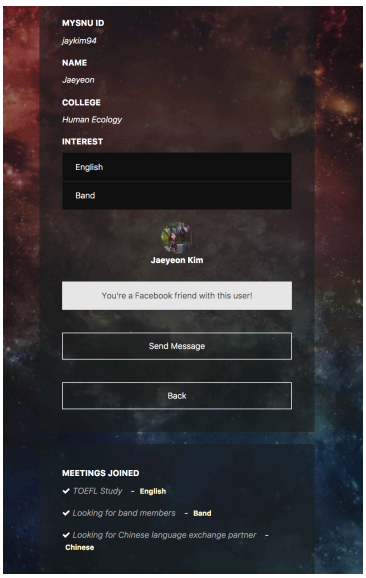
When user tries to access pages without sigining in



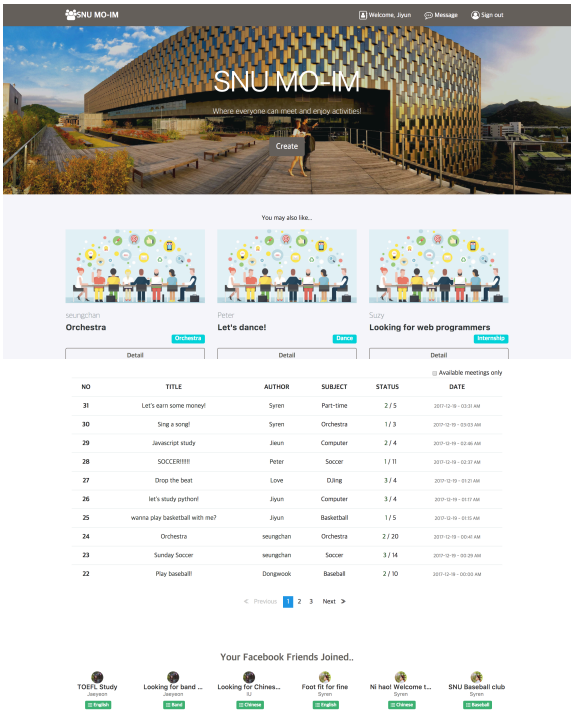
Sign Up(similar to Edit)

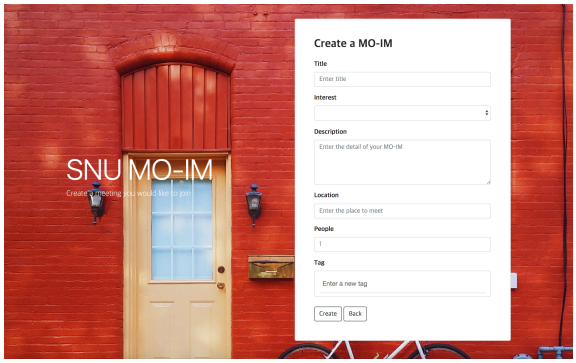


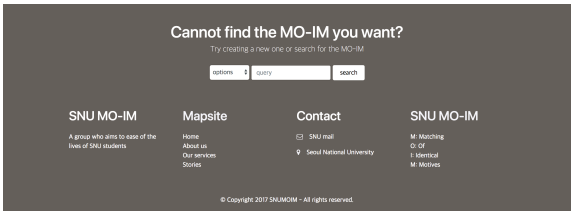
View Own Profile



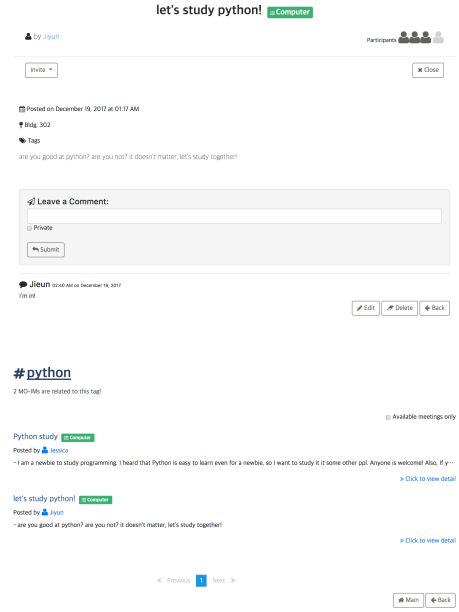
View Profile of Another User



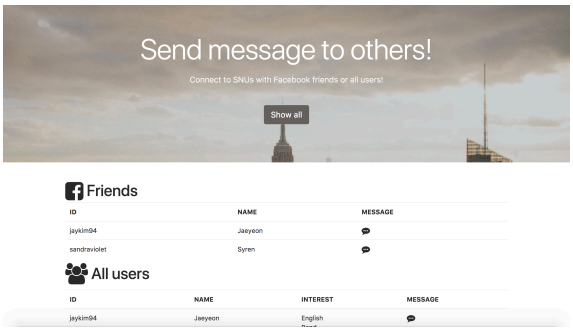




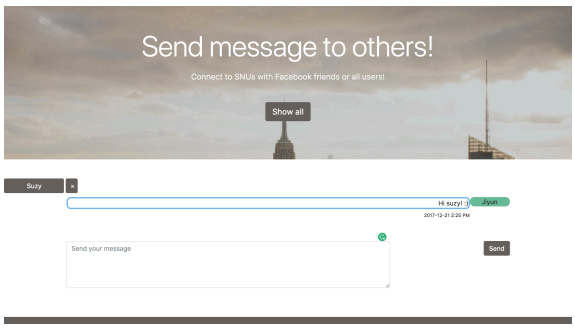
Meetings (Recommendation / List / Search)



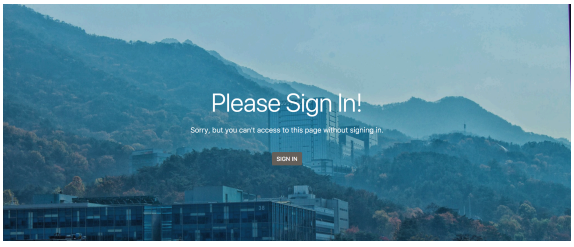
Create/Edit Meeting / Detail / Tag



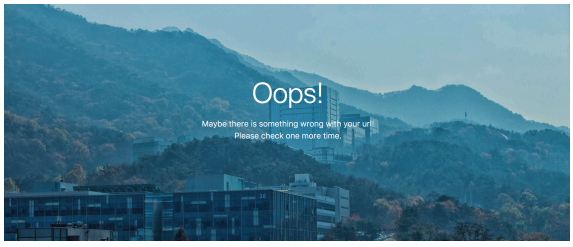
Messages (User List)



Message with a User



When user tries to access pages without sign in



when page not found