[주요한 필수 기능]

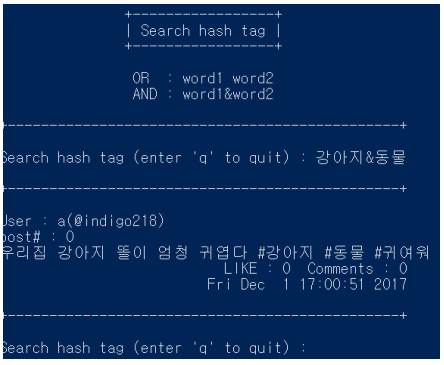
1. HASH tag 검색 기능

#로 시작하는 단어 단위로만 검색시키며 'text' 인덱스를 생성하여 search하게 하였기 때문에 굳이 태그의 리스트를 컬렉션의 필드에 추가할 필요성을 못 느껴서 추가하지는 않았다. 내가 검색하고자 하는 단어는 #를 붙여서 검색할 수도 있고(#여행) 또는 단어만(여행) 검색할 수도 있으며 두 경우 검색결과는 동일하게 하였다. 검색 과정은 다음과 같이 이루어진다. 우선 posts 콜렉션에 대해 기존의 인덱스는 모두 제거하고 content에 대한 text 인덱스를 생성하여 포스트의 본문에 대해 텍스트 검색이 가능하게 하였다. 그리고 나서 aggregate를 사용하여 tag로 입력한 단어가 들어간 문서를 검색하고 이를 다시 시간 순서대로 sort하여 cursor로 리턴하여 포스트들을 프린트하게 하였다.

```
db.posts.drop_indexes()
db.posts.create_index([('content',pymongo.TEXT)])
c=db.posts.aggregate([
    {'$match':{'$text':{'$search':tag}}},
    {'$sort': SON([('rawtime', -1)])}
])
post.printpost(db,c,'hashtag')
```

2. Hash tag AND OR 옵션 구현

그러나 단어 한가지만 검색하게 하는 것에서는 미진함을 느껴서 AND와 OR 옵션을 추가하였다. 이를 위해 공백으로 구분한 단어는 OR로, &로 구분한 단어는 AND라고 생각하였다. 강아지 동물" 이라고 검색하면 두 해시태그 중 하나라도 포함된 단어가 모두 나온다. 그러나 "강아지&동물"로 검색할 시에는 아래 그림처럼 두 해시태그 모두가 포함된 포스트만 나온다. Tag를 단일 단어가 아니라 2개 이상의 단어도 받을 수 있게 하였기 때문에 처리 과정이 더 복잡해졌다. 아래 코드를 보면 우선 공백으로 구분 가능한 모든 단어가 #로 시작하도록 변경한 후 &로 구분된 단어들의 양끝에는 ₩"를 붙여서 aggregate text search가 가능하게 만들었다.




```
if tag[0]!='#':
    tag='#'+tag
for i in range(len(tag)):
    if tag[i]==" ":
        if tag[i+1]!='#':
            tag=tag[:i+1]+"#"+tag[i+1:]
#"#aaa&#fff"->"\"#aaa\""
if "&" in tag:
    tag=tag.split("&")
    tag=list(map(lambda x: "\""+x+"\"",tag))
    tag=" ".join(tag)
```

[수정 사항]

원래는 post 컬렉션 내부에서 각 유저마다 시간순서대로 번호를 매겨서 직접 저장하였으나, 더 이상 그렇게 하지 않고 post들을 호출해서 print할 때마다 그때그때 순서를 매겨서 번호를 출력하게 하였다. 뉴스피드 상에서 좋아요, 댓글 구현하기 위해서는 그때그때 변하는 팔로잉 목록에 따

라 포스트가 나타나는 순서와 종류가 다이나믹하게 변하기 때문에 저장된 번호가 별로 쓸모가 없었기 때문이다. 아래 코드는 post들을 print하는 코드의 일부이며 c는 db.posts.find()에 대한cursor이다. P는 포스트의 번호를 나타내며 처음에는 0으로 초기화하고 루프를 돌때마다 1씩 올림으로써 각 포스트에 번호를 부여하고 그 번호를 프린트하게 된다. 좋아요할 포스트나 댓글 달 포스트를 선택할 때도 같은 방식으로 루프를 돌려서 선택하게 된다.

```
p=0
for apost in c:
    username=db.member.find_one({'_id':apost['uid']})['name']
    print("\n"+single)
    print('\nUser : ' + username+'(@'+apost['uid']+')')
    print('post# : ' + str(p))
    p+=1
```

[추가 기능]

1. 다른 회원 담벼락(wall) 찾아가기

기존 내 wall을 찾아가는 함수를 이용하였다. 원하는 사람의 id를 입력하면 그 사람의 담벼락에 찾아갈 수 있다. 이 메뉴를 통해 나 자신의 담벼락을 찾아갈 수도 있다.

```
              +-----------------+
              |     EXPLORE     |
              +-----------------+

+-----------------------------------------------+
You can explore the other user's wall!
Enter the id (q:quit) :
```

2.타인의 포스트에 대한 추가 기능

타인 혹은 나 자신의 포스트에 대해 다음과 같은 추가 기능 옵션을 부여하였다 : 좋아요, 좋아요 취소, 댓글보기, 댓글달기. 뉴스피드 등 포스트를 열람하는 기본 인터페이스에서 추가 옵션을 원하면 해당하는 번호를 입력하여 추가 옵션을 실행할 수 있다.

```
+-----------------------------------------------+

User : dd(@indigo218)
post# : 2
안녕하십니까
                        LIKE : 0  Comments : 0
                      Fri Dec  1 12:41:17 2017

+-----------------------------------------------+
1 좋아요 2 좋아요 취소 3 댓글 보기 4 댓글 달기 q 나가기
Enter :
```

2. 좋아요(LIKE) 기능

좋아요 기능을 구현하였다. 뉴스피드에서 내가 원하는 포스트의 번호를 누르면 좋아요 수를 올릴

수 있으며 취소할 수도 있다. Member 컬렉션에 'mylikes' 필드가 추가되었으며 이 필드에 내가 좋아하는 post컬렉션의 고유 _id가 어레이 형태로 저장된다. 한편 post 컬렉션에는 'like' 필드가 추가되어 숫자가 올라가게 된다. 좋아요 수는 뉴스피드나 담벼락에서 게시 시간 상단에 댓글 수와 함께 프린트되게 하였다. 한편 좋아요 취소시에는 posts 컬렉션의 like 숫자도 감소시키고 member 컬렉션의 mylikes 필드의 어레이에서 해당 포스트의 아이디를 삭제하였다. 마지막으로 exception처리는 다음과 같다

- 이미 좋아요를 누른 포스트에 다시 좋아요를 누를 수 없다
- 좋아요를 누르지 않은 포스트의 좋아요 취소를 할 수 없다

```
count=0
for apost in cursor:
    if count in post_numbers:
        unique_id=apost['_id']
        db.posts.update({'_id':unique_id},{'$inc':{'like':1}})
        db.member.update({'_id':user['_id']},{'$push':{'mylikes':unique_id}})
    count+=1
```

3. 댓글(comments) 기능

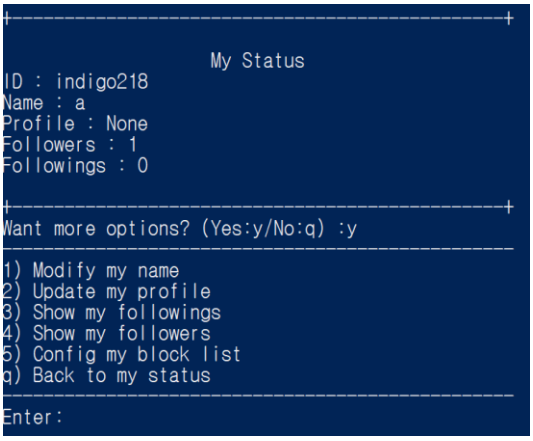
댓글 달기 기능을 구현하였다. Post 컬렉션에 'comments'필드를 추가하였으며 여기에는 어레이가 들어간다. 댓글을 달때마다 이 어레이에 dictionary형태의 comment element가 추가되고 {댓글다는 사람 아이디:00, 댓글내용:xxxxx} 형식으로 들어간다. 댓글보기를 선택하면 댓글을 확인하고 바로 아래에 새 댓글을 입력할 수 있으며 댓글을 입력하고 나면 해당 포스트에 대한 모든 댓글을 바로 확인할 수 있다.

```
1 좋아요 2 좋아요 취소 3 댓글 보기
Enter (q to quit) : 3
Enter post number : 0
```

```
+----------------------------------------+

User : a(@indigo218)
여행가자 여행 #여행 #배
                        LIKE : 0  Comments : 4
                      Fri Dec  1 14:47:20 2017



                     +댓글+
@indigo218 : 왜 댓글이 안써지지?
@indigo218 : 헐 써지네 ㅋㅋㅋ
@indigo218 : 아깐 왜 안됨?
@indigo218 : ㅋㅋㅋㅋㅋㅋㅋ

+----------------------------------------+
Enter comment :
```

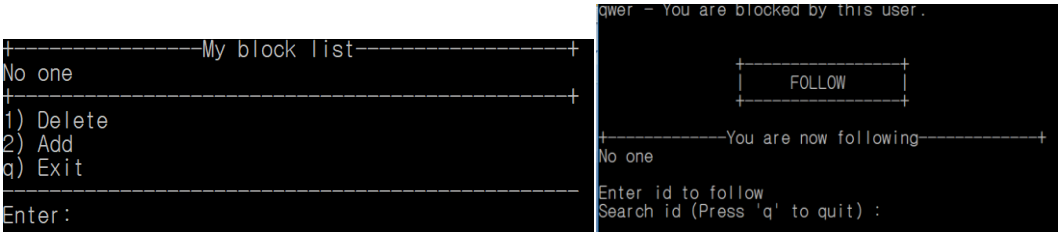
4. 회원정보 수정 기능

Status 창에 들어가면 이름 수정, 상태메시지 수정, 차단리스트 수정이 가능하게 하였으며, 여기서 follow 목록을 바로 들어갈 수도 있게끔 하였다.



5. 차단 기능

user콜렉션의 각 document는 block이라는 필드를 가졌으며 어레이로 되어있다. 이 어레이에는 내가 지정한 id들이 보관되며 이 id에 해당하는 회원들은 나를 더 이상 follow할 수 없게 된다. 아래 화면은 status 창에서 config my block 로 들어간 화면이다. 블록 리스트에 아이디를 삭제, 추가할 수 있음을 알 수 있다. 나를 차단한 사람을 팔로우하려고 하면 오른쪽과 같은 메시지가 뜨면서 팔로잉이 불가능하다.



6. os 모듈 사용

다음과 같이 os module을 import하고 system함수를 사용하여 새로운 메뉴로 들어갈 때마다 console창이 깨끗이 clean up되도록 조정하였다. Window의 경우에는 parameter가 'clr'로 우분투 등 나머지 os의 경우에는 'clear'가 들어가도록 설정해야 제대로 동작함을 알 수 있었다.

```
import os

def clear():
    os.system('cls' if os.name == 'nt' else 'clear')
```

[어려웠던 점]

1. 예외처리

가장 어려웠던 점은 예외처리하기였다. 우리가 SNS를 실행하면서 생각보다 엄청 다양한 경우의 수를 가진 행동을 하게 되어서, 개발자는 사용자의 그러한 다양한 행동에 일일이 대응해야만 했다. 예를 들어 존재하지 않는 아이디를 검색하거나 존재하지 않는 포스트를 삭제하려고 하는 경우 등 다양한 상황에 대해서 그때마다 적절한 지침을 내리는 것이 매우 복잡하였다.

2. 데이터베이스와 연동하기

Mongodb와 연동한 시스템이기 때문에 mongodb 언어를 구사하여 db의 데이터까지 조작해야 할 때 어려웠다. Pymongo 문법에 대한 지식이 부족하여 db 구성 자체를 뒤엎어버린 적도 여러 번 있었다. 예를 들어 댓글을 삭제하는 기능을 아직 구현하지 못하였는데, 이는 어레이 형태로 되어있는 필드에서 원하는 element만 삭제시키려면 어떻게 해야 하는지 막막했기 때문이다. 파이썬의 자료 구조 지식으로는 너무나 손쉽던 일도 mongodb를 조작하려고 하니 상당히 어려워졌다.

3. 보완점

최대한 단순화하려는 노력에도 불구하고 다양한 기능을 구현하고자 하는 의욕이 앞서다 보니 일부분에서는 인터페이스가 다소 복잡해진 측면이 있었는데, 사용자 편의성을 개선하기 위한 노력이 더 필요할 것 같다.

또한, 댓글 삭제 기능과 내가 좋아요를 누른 포스트 열람하기 등 당초 계획했으나 시간 관계상 완성하지 못한 기능이 아쉬웠다.