현성

1. 게시판 댓글 api

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/57781b1f-e41d-4a40-acc6-3bd9f088cd01/post_view.py

post_view의 278~304 라인

```
    comment_json_str=json.dumps(comment_json_object)

    if cur_post[0].comments is "":
        cur_post[0].comments=comment_json_str
        cur_post[0].save()
    else:
        cur_post[0].comments+="\n"+comment_json_str
        cur_post[0].save()


    print(cur_post[0].comments)
    comments_split=cur_post[0].comments.split("\n")
    comment_list=list()
    print(len(comments_split))



    for i in range(len(comments_split)):
        print(comments_split[0])
        print(type(comments_split[0]))
        cur_str=comments_split[i]
        cur_dict=eval(cur_str)
        comment_list.append(cur_dict)
    print(comment_list)
    res_post_json[0]["fields"]["comments"]=comment_list
    # return JsonResponse({"status": 200, "message": "comments in post", "data": "rkskek"})
return JsonResponse({"status": 200, "message": "comments in post", "data": res_post_json})
```

댓글 작성 api가 댓글을 작성했을 때 기존에 있던 모든 댓글과 현재 작성한 댓글을 모두 return 해 주어야 했는데, 기존 댓글들을 가져오면 json파일이 string타입으로 저장되어 있어 다시 댓글을 추가하고 dictionary 형태로 바꾼 후 list에 넣어서 json 파일로 변환하는 과정에서 어려움을 겪었습니다.

string 형태에 새 댓글을 추가할 때 이스케이프 문자("\n") 을 추가한 후 나중에 split() 함수를 통해 댓글들을 분리하였는데, 어떻게 했으면 더 좋았을지가 궁금합니다.

-> json 배열 형태로 저장하여서 추가/삭제 하는 방향이 어떨 까 싶습니다.

2. 매칭 시스템 api

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d70660bd-0edb-427c-b449-4b5f52865bfd/matching_view.py

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c6b18314-9950-4227-8465-0e232fffede7/party_matching_view.py

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/47f0a500-b689-41b2-be61-3b58d8850d43/matching_save_view.py

matching_view->party_matching_view->matching_saving_view 매칭 과정이 다음과 같이 3개의 서버로 이어지고 세부적으로는 다음과 같습니다.

(1) matching_view의 36~58번째 라인(레디스 큐 기반 대기열에 user rpush)

rd는 redis와 연결된 변수

```python
def enroll(request, pk):
    if request.method=="POST":
        access_token=request.headers.get('Authorization', None)
        ac=access_token

        user_info=requests.get("http://3.39.239.141:32513/auth/my", headers={'Authorization': ac})

        json_user_info=json.loads(user_info.content.decode('utf-8'))
        user_nickname=json_user_info["data"]["nickname"]
        # print(user_nickname)


        if User.objects.filter(username=user_nickname).exists() is False:
            user = User.objects.create_user(username=user_nickname)
        u = User.objects.get(username=user_nickname)
        # print(u)
        if not rd.exists(u.id):
            print("well")
            rd.set(u.id, pk)
            waiting_room_key="waiting_room_"+str(pk)

        # rd.rpush(waiting_room_key,u.username)
            rd.rpush(waiting_room_key,u.username)
```

(2) party_matching_view의 61~66번째 라인(레디스 큐 기반 대기열에 user lpop)

```python
for _ in range(4):
    cur_user_name=rd.lpop(cur_waiting_room).decode()
    print(_)

    cur_matching=Matching.objects.create(cur_party_id=cur_party, user_id=User.objects.get(username=cur_user_name))
    cur_matching.save()
```

(3) matching_save_view의 41~63번째 라인(pop된 user들을 json 파일로 저장 - k8s와 nfs사용예정)

cur은 mysql과 연결된 변수

```python
for index in range(4):
    party_id=rows[index][1]
    user_id=rows[index][2]
    get_raid_id="SELECT * FROM obj_create_party WHERE id="+str(party_id)
    k=cur.execute(get_raid_id)
    k_rows=cur.fetchall()
    raid_id=k_rows[0][3]

    get_raid_name="SELECT * FROM obj_create_raid WHERE id="+str(raid_id)
    s=cur.execute(get_raid_name)
    s_rows=cur.fetchall()
    cur_raid_name=s_rows[0][1]
    req_item_lev=s_rows[0][2]
    item_lev=req_item_lev

    username_list.append(user_id)
    raid_name=cur_raid_name
    p_id=party_id

    delete_data="DELETE FROM obj_create_matching WHERE user_id_id="+str(user_id)
    cur.execute(delete_data)
    # con.commit()
    cur.fetchall()
```

(4) REST API 서버에서 해당 유저 파티의 json 파일이 생성되었다면 그 json 파일을 return

party_matching과 matching_save 파일 같은 경우 while문을 통해 무한루프를 도는 상태에서 이벤트가 발생하면 코드를 수행하는 방식으로 이루어져 있는데, 게임의 매칭 시스템 구현도 이렇게 이루어 지는지, 더 개선된 방법은 무엇인지가 궁금합니다.

→ https://www.microsoft.com/en-us/research/wp-content/uploads/2006/10/Game-Developer-Feature-Article-Graepel-Herbrich.pdf