

post server api

팀명

객체 이름 및 구성 요소

```
class Post(models.Model):
    title=models.CharField(max_length=200, verbose_name='TITLE', help_text='*MAX 100*')
    author=models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts')
    content=models.TextField(verbose_name='CONTENT')
    published_date=models.DateTimeField(auto_now=True, verbose_name='PUBLISH_DATE')
    like = models.ManyToManyField(User, related_name='likes', blank=True)
    comments=models.TextField(verbose_name='comment')
    def __str__(self):
        return self.title

You, 2 days ago | 1 author (You)
class Comment(models.Model):
    post=models.ForeignKey(Post, on_delete=models.CASCADE)
    comment_author=models.ForeignKey(User, on_delete=models.CASCADE, related_name='comments')
    text=models.TextField(verbose_name='TEXT')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.text
```

Post 객체

- id: 객체 아이디(자동으로 생성)
- title: 게시글 제목
- author: 게시글 쓴 User에 대한 정보
- content: 게시글 내용
- published_date: 게시글이 생성된 시간
- like: 좋아요 수 (다대다)
- comments: 해당 게시글에 달린 댓글들

Comment 객체

- post: 해당 comment가 달리는 글(Post)에 대한 정보
- comment_author: 해당 comment를 작성한 User에 대한 정보
- text: 댓글 내용
- created_at: 댓글이 작성된 시간

id값은 어디에?

Quick example

This example model defines a **Person**, which has a **first_name** and **last_name**:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

first_name and **last_name** are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column.

The above **Person** model would create a database table like this:

```
CREATE TABLE myapp_person (
    "id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Some technical notes:

- The name of the table, **myapp_person**, is automatically derived from some model metadata but can be overridden. See [Table names](#) for more details.
- An **id** field is added automatically, but this behavior can be overridden. See [Automatic primary key fields](#).
- The **CREATE TABLE** SQL in this example is formatted using PostgreSQL syntax, but it's worth noting Django uses SQL tailored to the database backend specified in your [settings file](#).

“An id field id added automatically” 부분을 참조한다.

django의 models에서 선언한 클래스로 객체를 생성하면, bigint의 id가 자동으로 부여된다.

자세한 내용 참조 → <https://docs.djangoproject.com/en/4.1/topics/db/models/>

기능 별 url

1. Post

- /post/ : 전체 게시판 글이 보이는 곳 (시간 역순)

커뮤니티 게시판

No	TITLE	AUTHOR	CONTENT	PUBLISHED_DATE	MANAGEMENT
10	It is Title	user1234	contentcontent	2023-02-03	수정 삭제
9	It is Title	user1234	contentcontent	2023-02-03	수정 삭제
8	It is Title	user1234	contentcontent	2023-02-03	수정 삭제
7	helloq	user1234	test contentw	2023-02-03	수정 삭제
6	helloq	user1234	test contentw	2023-02-03	수정 삭제
5	helloq	user1234	test contentw	2023-02-03	수정 삭제
4	helloq	user1234	test contentw	2023-02-03	수정 삭제
3	helloq	user1234	test contentw	2023-02-03	수정 삭제
2	hello	user1234	test content	2023-02-03	수정 삭제
1	페이지 처리-57		페이지 처리 데이터 입력_57	2023-01-30	수정 삭제

코드는 어떻게 구성? django의 model로 만든 객체들을 json직렬화 하여 보여준다.

```
def index(request):
    You, last week * 게시판 글 등록 수정 삭제 목록 기능 구현 ...
    board_list=Post.objects.all().order_by('-id')
    print(type(board_list))
    board_list_json = serializers.serialize("json", board_list)
    print(board_list_json)

    return JsonResponse({"code": 200, "message": "All Posts look successful", "data": board_list_json})
```

해당 api를 호출하면 다음과 같이 title, author, content등의 정보가 출력된다.

GET http://127.0.0.1:8000/post/ Send

Params Authorization Headers (11) Body **●** Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 336 ms Size: 18.44 KB Save Response

Pretty Raw Preview Visualize JSON ⌵ ⌵

```

1  {
2    "code": 200,
3    "message": "All Posts",
4    "data": [
5      {
6        "model": "post.post",
7        "pk": 76,
8        "fields": {
9          "title": "0208 title",
10         "author": "10",
11         "content": "02081115 content",
12         "published_date": "2023-02-08T05:58:48.742Z",
13         "comments": "",
14         "like": [
15           10
16         ]
17       }
18     ],
19   },
20 }
```

```

"data": {
  "total": number,
  "posts": [
    {
      "model": "post.post",
      "pk": 76,
      ...
    }
    ...
  ]
}
```

pk값 또한 자동으로 부여되는 값으로, 자동으로 id값이 부여된다.

즉 django의 model로부터 생성한 객체는, id값과 pk값에 별도의 설정을 하지 않는다면 동일하다.

Automatic primary key fields

By default, Django gives each model the following field:

```
id = models.AutoField(primary_key=True)
```

This is an auto-incrementing primary key.

If you'd like to specify a custom primary key, just specify `primary_key=True` on one of your fields. If Django sees you've explicitly set `Field.primary_key`, it won't add the automatic `id` column.

Each model requires exactly one field to have `primary_key=True` (either explicitly declared or automatically added).

더 자세한 내용은 해당 링크를 참조한다.

<https://docs.djangoproject.com/en/1.11/topics/db/models/#automatic-primary-key-fields>

실제로 다음과 같이 임의의 api에서 코드를 작성하고 api호출 후 출력을 확인해 보면 같다는 것을 확인할 수 있다.

```
def detail(request, pk):
    board_list=get_object_or_404(Post, id=pk)
    # print(board_list.like.all())
    print(board_list.id)
    print(board_list.pk)
    return JsonResponse({'status': 'ok'})
```

```
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
75
75
[06/Feb/2023 16:40:16] "GET /post/detail/75/ HTTP/1.1" 200 16
```

- /post/?page=n (n은 1이상의 정수, 다른 api들과 달리 url끝에 /가 없음 주의)
 - n번째 페이지들의 게시글들을 한 페이지당 10개씩 json 형태로 반환

```
def index(request):
    page = int(request.GET.get("page", 1) or 1)
    page_size = 10
    limit = int(page_size * page)
    offset = int(limit - page_size)
    board_list=Post.objects.all().order_by('-id')[offset:limit]
    page_count = ceil(Post.objects.count() / page_size)
    board_list_json = serializers.serialize("json", board_list)
    res_post_json=json.loads(board_list_json)

    return JsonResponse({"code": 200, "message": "All Posts", "data": res_post_json})
```

- 최신 순 정렬을 위해 게시글의 id값을 역순으로 정렬 후 슬라이싱
- 페이지가 1인 경우(게시글의 id값 15~6)

GET ▼ http://3.35.71.51:8000/post/?page=1

Params ● Authorization ● Headers (10) Body ● Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DE
<input checked="" type="checkbox"/>	page	1	
	Key	Value	De

Body Cookies Headers (8) Test Results ⌚ Status: .

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "code": 200,
3   "message": "All Posts",
4   "data": [
5     {
6       "model": "post.post",
7       "pk": 15,
8       "fields": {
9         "title": "title15",
10        "author": 1,
11        "content": "content15",
12        "published_date": "2023-02-12T06:54:06.811Z",
13        "comments": "",
14        "like": []
15      }
16    },
17    {
```

- 페이지가 2인 경우(게시글의 id값 5~)

GET http://3.35.71.51:8000/post/?page=2

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DE
<input checked="" type="checkbox"/>	page	2	
	Key	Value	De

Body Cookies Headers (8) Test Results Status:

Pretty Raw Preview Visualize JSON

```

1  {
2    "code": 200,
3    "message": "All Posts",
4    "data": [
5      {
6        "model": "post.post",
7        "pk": 5,
8        "fields": {
9          "title": "title5",
10         "author": 1,
11         "content": "content5",
12         "published_date": "2023-02-12T06:53:28.930Z",
13         "comments": "",
14         "like": []
15       }
16     ],
17   }

```

- /post/regist/ : 글 등록 버튼 클릭 시 글 등록 폼으로 이동
 - 글 등록 폼에서 사용자가 글과 제목을 작성하고 글 등록 클릭 시 header로 access token을 달고, 인증 서버의 /auth/my 라는 api를 호출하여 지금 사용자가 등록된 유저 인지 확인

```

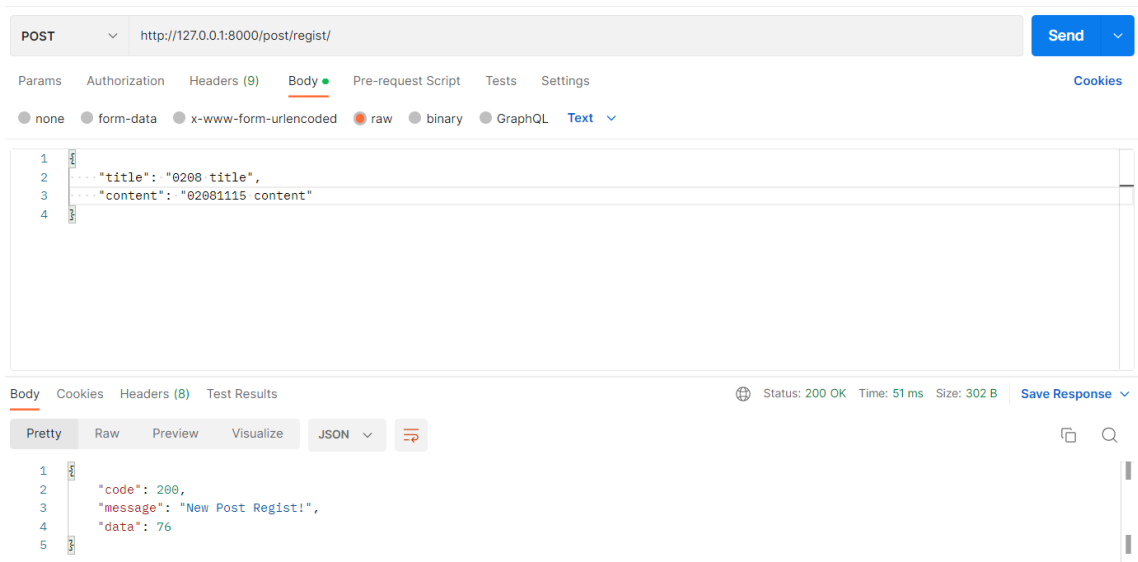
access_token=request.headers.get('Authorization', None)
ac=access_token
user_info=requests.get("http://192.168.195.15:8080/auth/my", headers={'Authorization': "Bearer "+ac})
json_user_info=json.loads(user_info.content.decode('utf-8'))

```

등록된 유저라면 다음 단계의 프로세스 계속해서 진행

- 글 등록 폼 프론트 서버에서 보내는 json 파일에 담겨있어야 하는 내용
 - titile(글 제목)
 - content(글 내용)
- 다음 과정을 헤더에 액세스 토큰을 넣고 진행한다. (테스트 단계에서는 임의의 User객체를 생성하고 access token을 임의로 만든 후 디코딩 과정을 거쳤기에 Bearer를 진행하지 않아도 됨)

- 이제 실제로 User가 글을 등록한 것처럼 body에 title과 content를 넣은 뒤 /post/regist/로 POST 한다.



data에 76이라는 결과값이 나오는데 이는 선언한 Post객체의 id 값을 가리키며, 실제 코드에서는 다음과 같은 값을 return 하도록 하였다.

```
new_post=Post.objects.create(author=u, title=body["title"], content=body["content"])
# print(new_post)
# return JsonResponse({'message': new_post.id})
return JsonResponse({"code": 200, "message": "New Post Regist!", "data": new_post.id})
```

Post라는 객체들 중 id값이 76번인 객체가 실제로 존재하는지 mysql DB를 확인해본다.

Query 1 x

Limit to 1000 rows

```

1 • SHOW DATABASES;
2 • USE board;
3 • SHOW TABLES;
4 • ALTER TABLE post_post ADD comments TEXT;
5 • SELECT * FROM post_post;

```

Result Grid

	id	title	author_id	content	published_date	comments
▶	76	0208 title	10	02081115 content	2023-02-08 02:15:...	
	75	0205 title	10	02052136 content	2023-02-08 02:05:...	{'cur_user_id': 10, 'cur_user_co...

앞서 body에 넣었던 0208 title을 title로 가지고, 02081115 content를 content로 가지는 Post 객체가 id 값 76를 가지면서 post_post 테이블에 저장되어 있는 것을 확인할 수 있다.

- /post/edit/<int:pk>/
 - 수정하려는 User의 id값과, Post 객체 id가 pk인 값의 Post객체에 있는 author_id 값이 같다면, 해당 Post객체의 title과 content내용을 overwrite한다.
 - 글 수정 폼 프론트 서버에서 보내는 json 파일에 담겨있어야 하는 내용
 - title
 - content
 - edit 기능을 확인하기 위해 다음과 같이 id값이 76인 post를 가져온다.

GET http://127.0.0.1:8000/post/ Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings [Cooki](#)

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk E
Key	Value	Description		

Body Cookies Headers (8) Test Results [Status: 200 OK](#) [Time: 336 ms](#) [Size: 18.44 KB](#) [Save Response](#)

Pretty Raw Preview Visualize JSON

```

1  {
2    "code": 200,
3    "message": "All Posts",
4    "data": [
5      {
6        "model": "post.post",
7        "pk": 76,
8        "fields": {
9          "title": "0208 title",
10         "author": "10",
11         "content": "02081115 content",
12         "published_date": "2023-02-08T05:58:48.742Z",
13         "comments": "",
14         "like": [
15           10
16         ]
17       }
18     ],
19   }

```

id값이 76인 post에 대해 edit api를 실행하면

POST http://127.0.0.1:8000/post/edit/76/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL [Text](#)

```

1  {
2    "title": "0208 title EDIT",
3    "content": "02081115 content EDIT"
4  }

```

Body Cookies Headers (8) Test Results [Status: 2](#)

Pretty Raw Preview Visualize JSON

```

1  {
2    "code": 200,
3    "message": "Post Edited",
4    "data": [
5      {
6        "model": "post.post",
7        "pk": 76,
8        "fields": {
9          "title": "0208 title EDIT",
10         "author": "10",
11         "content": "02081115 content EDIT",
12         "published_date": "2023-02-08T06:08:43.748Z",

```

수정된 내용이 쓰여지게 된다. 다시 모든 post 객체들을 불러와 확인해 본다.

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/post/`. The response status is 200 OK, with a time of 203 ms and a size of 18.45 KB. The response body is displayed in JSON format, showing an array of posts. The first post in the array has a title of "0208 title EDIT", author "10", and content "02081115 content EDIT".

```
1 {
2   "code": 200,
3   "message": "All Posts",
4   "data": [
5     {
6       "model": "post.post",
7       "pk": 76,
8       "fields": {
9         "title": "0208 title EDIT",
10        "author": "10",
11        "content": "02081115 content EDIT",
12        "published_date": "2023-02-08T06:08:43.748Z",
13        "comments": "",
14        "like": [
15          10
16        ]
17      }
18    },
19    ...
20  ]
21 }
```

수정된 사항이 반영된 것을 확인할 수 있다.

- `/post/delete/<int:pk>`
 - edit과 동일하게 User 인증 후 DB에서 삭제하는 과정을 거친다.

POST ▼ http://127.0.0.1:8000/post/delete/77/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **Text** ▼

```

1  [
2    {
3      "title": "0208 title EDIT",
4      "content": "02081115 content EDIT"
5    }
6  ]

```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▼ ↺

```

1  {
2    "code": 200,
3    "message": "Post Deleted",
4    "data": null
5  }

```

해당 api를 호출하여(body 내용은 무시한다) Post객체를 삭제하고, 삭제 되었는지 확인한다.

Limit to 1000 rows

```

1 • SHOW DATABASES;
2 • USE board;
3 • SHOW TABLES;
4 • ALTER TABLE post_post ADD comments TEXT;
5 • SELECT * FROM post_post;

```

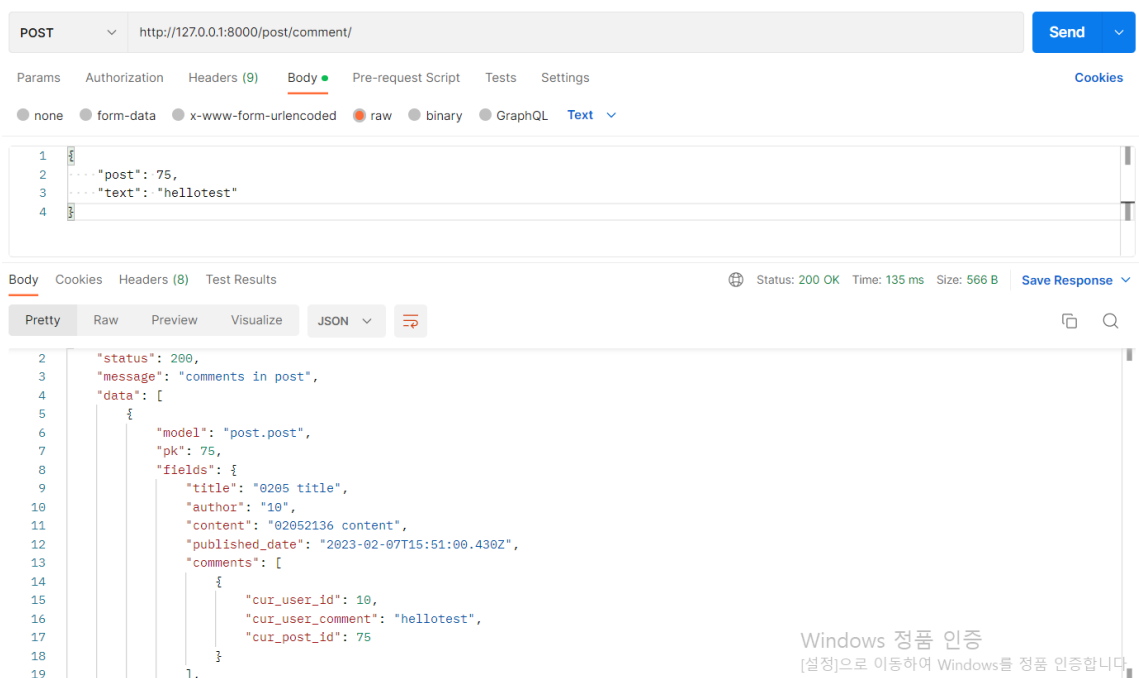
id	title	author_id	content	published_date	comments
75	0205 title	10	02052136 content	2023-02-08 02:05:...	{'cur_user_id': 10, 'cur_user_co...
74	0204 title	10	02041616 content	2023-02-07 15:51:...	NULL
73	It is Title	10	contentcontent	2023-02-03 03:44:...	NULL
72	It is Title	10	contentcontent	2023-02-03 03:44:...	NULL
71	It is Title	10	contentcontent	2023-02-03 03:38:...	NULL
70	helloq	10	test contentw	2023-02-03 02:34:...	NULL

77번째 객체가 보이지 않는 것을 확인할 수 있다.

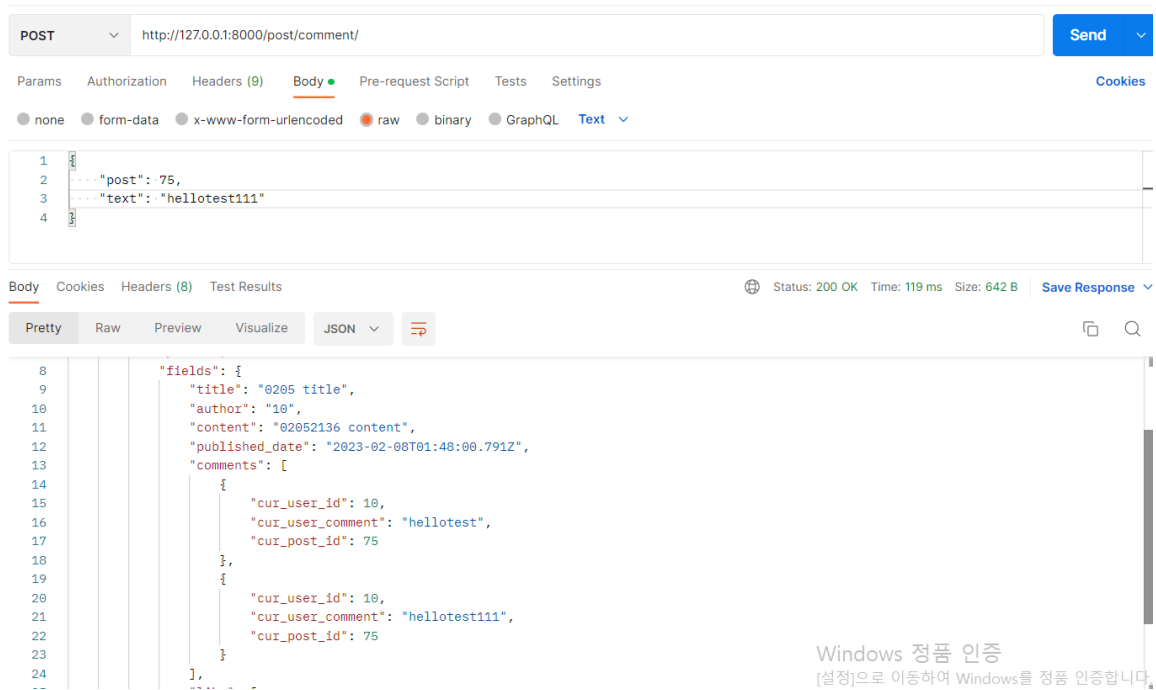
2. Comment

- /post/comment/

- 댓글 작성 시 호출되는 api로, api 호출 시 user가 post id 값과 댓글내용(text)를 넣어 보내면,
해당 post id 값을 갖는 글에 작성된 모든 댓글에 현재 작성된 댓글내용, 댓글 작성자, 게시글 id를 추가해서 보여준다.
- 다음과 같이 post id값과 댓글내용을 넣어 보내면 comments 배열에 현재 댓글을 작성한 user의 id정보, user가 작성한 댓글, 댓글을 작성한 게시물의 id가 담겨있는 것을
확인할 수 있다.



다른 댓글 내용으로 api를 한 번 더 호출해 본다.



comments 배열에 추가되는 것을 확인할 수 있다.

1. Like

- /post/like/<int:pk>/
 - 게시물(Post객체)의 추천 버튼을 누르면 호출되는 api로, api 호출 시 header에 담긴 access token을 바탕으로 user정보를 알아낸 후 해당 user가 Post객체의 like 변수에 이미 존재한다면 해당 user를 지우고 존재하지 않는다면 Post객체의 like 변수에 user를 추가한다.
 - GET 방식으로 호출한다.

GET http://127.0.0.1:8000/post/like/78/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL Text

```

1 {
2   ... "title": "02081607 title",
3   ... "content": "02081607 content"
4 }

```

Body Cookies Headers (8) Test Results Status: 200 OK Time: 37 ms Size: 684 B Save Res

Pretty Raw Preview Visualize JSON

```

1 {
2   "code": 200,
3   "message": "Post Like Added",
4   "data": [
5     {
6       "model": "auth.user",
7       "pk": 10,
8       "fields": {
9         "password": "pbkdf2_sha256$260000$cDh3nCydX67onmrvqg4Gj8$i2rgwYCsIw6NxvdYNTT02NTBU5FAV1fiqzW56XvTrj4=",
10        "last_login": null,
11        "is_superuser": false,
12        "username": "user1234",
13        "first_name": "",
14        "last_name": "",
15        "email": "",
16        "is_staff": false,
17        "is_active": true,
18        "date_joined": "2023-02-02T17:16:28.114Z",

```

Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증

75번째 Post 객체에 Like api를 실행시키고, 다음과 같이 75번째 Post 객체의 like 항목을 출력한다.

```

Django version 3.2, using settings 'loatus_board.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
<QuerySet [<User: user1234>]>

```

user1234라는 user 정보가 나오는 것을 확인할 수 있고, Like api를 한 번 더 실행시키면 user정보가 없어지는지 확인한다.

```

if u in post.like.all():
    post.like.remove(u)
    cur_post_json = serializers.serialize("json", post.like.all())
    res_post_json=json.loads(cur_post_json)
    return JsonResponse({"code": 200, "message": "Post Like removed", "data": res_post_json})
else:
    post.like.add(u)
    cur_post_json = serializers.serialize("json", post.like.all())
    res_post_json=json.loads(cur_post_json)
    return JsonResponse({"code": 200, "message": "Post Like Added", "data": res_post_json})

```

Post 객체의 해당 user의 like 여부를 판단하는 코드 앞 뒤에 Post 객체의 like에 대해 출력해보면,

(u는 해당 api를 호출한 user로, access token을 통해 판단)

```

<QuerySet [ <User: user1234> ]>
<QuerySet []>
[06/Feb/2023 10:13:31] "GET /post/like/75/ HTTP/1.1" 200 16

```

user 정보가 사라진 것을 확인할 수 있다.