-auth의 views에 PasswordChange와 관련된 View,Form이 있다 이를 사용해 비밀 번호 변경 페이지 구현가능

-포스팅 쓰기는 instagram app에서 구현(생성함), 사진 한장과 caption,tag 만들기

-HTML에서 url reverse사용시 {% url '앱이름:view 이름 % '} 이렇게 하면 되는듯?

-Model을 만들고 나면 Admin.py에서 @admin.register를 사용해 admin 단에서 확인이 가능하다.

-re.findall(r"#[a-zA-Z\dㄱ-힇]+", post.caption) 이거 정규 표현식인데 #이들어간 a-z 즉 모든 알파뱃과 한글을 post.caption에서 가져온다란 의미이다.

===

<User Page 만들기>

-이미지를 동그랗게 만들기 위해 getbootstarp.com의 문서에서 검색 가능

-img의 class에 "rounded-circle" 추가하면된다.

-User Page의 핵심 ==> url이다 정규 표현식을 통해 username과 관련된 url이 들어와야 이동시키게 하는것이 중요하다.

re\_path(r'^(?P<username>[\w.@+-]+)/$',views.user\_page, name='user\_page') 이렇게 하면된다.

-중요한거는 $ 부분인데 달러표시를 해줘야 username만 들어왔을때 동작을 하게 한다.

======================

-API 서버? ==>앱/웹 서비스를 만드는 개발자들이 이용하는 data 위주의 서비스 시간이 지나도 호환성 유지가 필요로 한다.

데이터는 주로 json을 사용하며(모든건 x) 웹 서비스를 이용하는 경우엔 항상 최신버전을 사용

-REST==> Representational State Transfer의 약자로 대표 상태 전달? 이란 느낌으로 해석하면 된다.

아키텍처 스타일을 REST라고 부르며, 프로토콜에 독립적이다. 일반적인 REST 구현에서는 HTTP를 사용!

-RESTful API 디자인 원칙(by MS Azure 아키텍처 디자인)

1)리소스 중심으로 디자인(리소스? ==> 장고 모델 같은것)

2)클라이언트에서 액세스할 수 있는 모든 종류의 개체/서비스가 리소스에 포함

3)리소스마다 해당 리소스를 고유하게 식별하는 식별자

4)요청/응답 format으로 주로 json을 사용한다

5)균일한 인터페이스를 적용, 리소스 표준 HTTP 동사(GET,PUT등)를 사용한다

-리소스 중심으로 API 구성! 심플한 URI! 가중요하다

-기본 HTTP Method 기준으로 작업의 정의?

1)GET: 리소스의 표현, 응답 본문의 리소스의 세부정보를 보여준다

2)POST: 새 리소스 생성 요청, 응답 본문에 새 리소스 세부정보를 제공(멱등성 미보장)

즉 POST하면서 세부정보 제공 필요!

3)PUT: 기본 리소스를 대체(업데이트 개념), 요청 본문에 갱싱할 정보 제공(멱등성 보장되야됨)

멱등성?==>몇번을 하든 서버상에서는 같은 결과가 보장되는 개념?

4)PATCH: 기본 리소스를 부분 대체한다. (멱등성 미보장)

5)DELETE: 지정 리소스 제거

-Content-Type 해더:요청시 처리를 원하는 형식 지정하면 서버에서는 형식대로 응답한다. 서버측에서 형식을 지원하지 않으면 415 코드 반환

-HTTP METHOD 별 다양한 상태 코드

1)GET: 성공시 200, 리소스 못찾으면 404

2)POST: 성공시201응답, 새 리소스의 URI는 응답의 Location 해더로, 200,204도 반환할 수 있다 잘못된

데이터 요청시 400에러!

3)PUT: 기존 리소스를 업데이트할 경우 200(OK)또는 204(내용없음) 반환, 상황에 따라 업데이트할 수 없는 경우 409 충돌 반환

4)DELETE: 성공시 204응답(내용없음) 리소스가 없는 경우 404 응답

5)비동기 작업: 작업 완료에 시간이 오래 걸릴 경우 다른 TASK Queue를 통해 비동기 처리 할 수 있다. 이 때 요청은 수락되었지만 완료되지 않았는 경우 202 응답(수락됨 응답)

-Django-Rest\_framework: 장고가 아닌 별도의 third-party 애플리케이션이다. 즉 장고 API에서 널리 쓰여지기 때문에공부하도록 한다!

즉 REST API 컨셉을 만들 수 있게 도와주는게 DRF이다.

<DRF의 주요 기능들>

1)Serializer/ModelSerializer를 통한 데이터 유효성 검증및 데이터 직렬화

2)각종 Parser를 통한 데이터 처리

3)APIView/Generic/ViewSet/ModelViewSets를 통한 요청 처리

4)Renderer를 통한 응답 포맷 지원

5)인증 및 권한 체계==> JWT지원

6)Thorottling: 최대 호풀 횟수 제한

-CRUD란? ==> C는 Create(새 레코드 생성), R은 Read/Rerieve(레코드 목록 조회), U는 Update(특정 레코드 수정), D는 Delete(특정 레코드 삭제)란 뜻이다

CRUD는 리소스에 대한 대표적 동작일뿐 API의 전부는 아니다.

=====

-한 post model에 대한 API 서비스 제공시 URL 설계?

1)새 포스팅 내용을 받아 등록하고 확인 응답-->/post/new/ 주소로 POST 요청

2)포스팅 목록 및 검색응답-->/post/ 주소로 GET 요청

3)10번 포스팅 내용 응답--> /post/10 주소로 GET 요청

-REST API로 설계시 5개의 뷰 처리가 필요로 한다.

컨셉 코드?

-view.py에서 JsonResponse를 통해 응답을 준다(render가 아니고!)

===============

<다양한 HTTP 클라이언트 프로그램>

-GUI로는 postman(Powerful API Client)가 있고 CLI 프로그램은 HTTPie, cURL이 있다. 라이브러리는 request를 사용한다.

-여기서는 HTTPie를 사용한다. pip로 다운 받고 'http (원하는 url)'을 입력하면 해당 url의 JSON응답을 받는다.

-HTTP 테스트용 서버는 httpbin.org이다!

-JSON으로는 POST를 받을 수 없고 body에 unicode로 메시지가 전달되고 --form을 사용하면 request.post에 메시지가 들어온다.

=============

<JSON 직렬화>

-직렬화란? ==>프로그래밍 언어의 통신에서 데이터는 필히 문자열로 표현되어야함

즉 직렬화란 송신자가 객체를 문자열로 변환하는 작업을 말한다(Serialize)/ 수신자는 비직렬화를 한다.

따라서 직렬화에 대한 비 직렬화가 있어야 한다!

-공통으로 지원하는 직렬화 포맷은 JSON,XML이 있고 특정 언어에서만 지원하는 포맷이 있다(파이썬은 Pickle이 있다)

-보통의 웹 세상에서는 GET에 대해 HTML 포멧으로 응답, POST에 대해서는 application/x-www-form-urlencoded 인코딩 또는 multipart/form-data 인코딩으로 요청하고 HTML 포맷으로 응답

API 서버에서는 json 인코딩된 요청/응답을 한다!

-JSON은 **다른 언어/플랫폼과 통신할때 주로 사용한다! (자바랑 통신도 가능)**, 표준 라이브러리 json제공을 한다(즉 파이썬 기본 데이터 타입에 대해서는 지원하지만 장고 모델or 우리가 직접 만든 model에 대해서는 직렬화를 직접 커스텀 해야된다. 여기가 중요한 부분임)

JSON을 import하고 **json.dumps()**로 직렬화 한다

-pickle에서는 pickle.dumps로 하면 된다. (저장 타입은 Byte 타입이다.)

-핵심은 말했듯이 **JSON/Pickle 둘 다 “파이썬” 라이브러리지 “Django”라이브러리가 아니기 때문에 추가적인 Rule 설정이 필요로 하다.**

-추가적인 Rule은 DjangoJSONEncoder를 import해주면 된다. (from Django.core.serializers.json) 하지만 이 경우도 오류가 발생할 수 있다

🡺해결 방법 중 제일 간단한 거는 그냥 직접 변환 때리는 방법이 있다.

Data=[{‘id’: post.id, ‘title’: post.title, ‘content’: post.content} for post in Post.objects.all()] 이렇게!

-json.dumps의 **ensure\_ascii** 옵션, 기본은 True 값이고 이는 한글 문자열을 unicode로 바꿔준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

🡪\\u라는글자가 유니코드라는 뜻

-근데 위의 방식대로 직접 하는 것은 매우 귀찮다. 따라서 **나만의 Rule을 class로 커스텀하는 방법**이 좋다.

1)DjangoJSONEncoder를 상속받는 클래스 만들기

2) default라는 함수 하나만들고 obj 인자를 받는다 obj인자는 객체를 어떻게 변환하는지에 관한 인자이다.

텍스트이(가) 표시된 사진

자동 생성된 설명 <예시>

위의 예시에서 if문으로 obj가 QuerySet이면 튜플로 바꾸고 재귀적으로 다시 올려 보내다. 뭐 이런 느낌이다 if문은 내 마음대로 커스텀 할 수 있다.

-\*\*최종적으로 **json.dumps(직렬화 할 내용, cls=”Json 엔코더”, ensure\_ascii=)** 이렇게 구현한다

<DRF에서 지원하는 **rest\_framework.renderer.JSONRender**>

-장고의 DjangoJSONEncoder를 상속받지 않고 기본 json.JSONEncoder를 상속받아 구현한다.

-**Model 타입은 미지원**🡪ModelSerializer를 통해 변환해라

<DRF에서 지원하는 **rest\_framework.renderer.JSONRenderer**>(er이 붙음)

-json.dumps에 대한 래핑 클래스로 보다 편리한 직렬화 지원

-utf-8인코딩도 추가 수행

data = Post.objects.all() JSONRenderer().render(data) 이렇게 사용

-당연히 모델에 대한 변환은 되지 않는다.

**<장고 Form&ModleForm VS DRF Serializer/ModelSerializer>**

-장고의 form과 DRF의 Serializer는 서로 유사하다. 코드 구성도 그렇고 둘 다 Form 필드 지정 또는 Model로부터 데이터를 읽어 오기 때문+ 유효성 검사 및 획득도 한다.

-다만 차이점은 장고는 Form 태그가 포함된 HTML을 생성하지만 DRF는 Form데이터가 포함된 JSON 문자열을 생성한다.

-ModelSerializer를 사용할 때 Post.objects.all()처럼 쿼리 set을 넘겨준다면 Many=True 옵션을 넣어주고 하나의 쿼리만 넘겨준다면 그냥 사용해도 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 사진이 예시다

-여기서 .data의 타입은 **ReturnDict** 타입이다.

<ReturnDict 타입?>

-Serializer.data의 데이터 타입으로 OrderedDict를 상속받으며 serializer를 키워드 인자로 삼는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-위처럼 만들어진 serializer.data는 json.dumps or JSONRenderer로 직렬화가 가능하다. WHY? 🡺객체이기 때문에! 파이썬 기본 자료 구조이다.

**<장고 View에서의 JSON 응답>**

-View는 기본적으로 **HttpResponse 타입의 응답을 필요로 한다**. 2가지 방법이 있다

1)직접 json.dumps를 통해 직렬화 된 문자열을 획득하여 HttpResponse를 통해 응답

2)1번을 정리한 **JsonResponse** 지원🡪내부적으로 json.dumps를 사용해 DjangoJSONEncoder가 디폴트 지정되어 있다.

즉 2번을 사용하면 DjangoJSONEncoder에서만 지원하는 옵션을 위주로 사용해라

-DjangoJSONEncoder를 상속받은 **커스텀 Rule 인코더로도 Queryset을 직렬화 가능**하다.

**<DRF를 통한 JSON 응답>**

-DRF Response 활용하는 방법

From res\_framework.response import Response를 import해 Response(serializer.data)로 JSON 직렬화가 가능하다.



-Response와 APIView: 모든 DRF의 뷰는 APIView를 상속받고 APIView를 통해 Response에 다양한 속성이 지정됨 🡺Response는 API View를 달고 산다.

**<실제 활용 예제>**

-간결하게 사용한다

From rest\_framework import **generics**

Class PostListAPIView(generics.**ListAPIView**): #이건 List기능만 지원한다.

Qs=post.objects.all()

Serializer\_class=PostModelSerializer

Post\_list=PostListAPIView.as\_view()

-포스팅 조회 응답에 username을 포함시키려면?

1) serializer.py에 가서serializers.ReadOnlyField의 source=’author.username’ 옵션을 추가 당연히 models에 author는 추가해주기

2)중첩된 serializer를 통해 구현하기

텍스트이(가) 표시된 사진

자동 생성된 설명

**<APIView, JSON 응답뷰>**

**<Serializer를 통한 View 처리>**

-Form과 유사하지만 Form은 인자로 data를 처음으로 받지만 Serializer는 **instance**를 받는다.

-data를 넘기고 싶다면 PostSerializer(data=request.POST) 같이 사용

-DRF는 **APIView**를 상속받아서 Class 기반 View를 작성하고, 함수 기반 View는 **@api\_view** 장식자를 받아서 작성해 준다.

-APIView에는 쿼리셋, seializer\_class 등을 설정할 수 있다.

-@장식자에는 APIView의 상속받아 로직을 처리한다. 🡺결국 APIView Class 활용!

\*\*결론은 해당 APIView에는 다양한 기본속성을 부여하는 기능이 내장되어 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-renderer\_classes에는 JSONRenderer와 TemplateHTMLRenderer가 있다.

-parser에서는 JSONParser와 FormParser, MultipartParser가 있다.

-인증 클래스에는 세션에 기반한 인증, HTTP Basic인증이 있으며 throttle클래스로 빈 튜플을 만들 수 있다.

-permission\_classes로 AllowAny를 설정하면 누구라도 접근 허용하게 해준다

-content\_negotiation\_classes로 같은 URL의 요청에서 JSON/HTML 응답을 요구하는지 판단한다.

-versioning\_class로 API 버전 정보를 탐지할지, 아니면 해당 버전의 API\_View를 호출되도록하는지 설정한다.

**<APIVIew와 @api\_view>**

-DRF의 2가지 기본뷰로 **APIView**와 **@api\_view**가 있다 차이점은 CBV용인지 FBV인지 차이임

-APIView는 하나의 CBV🡪하나의 URL만 처리 가능하다. 각 method에 맞게 멤버함수를 구현 가능 즉 각 method에 대한 action을 만들어 줄 수 있다. 순서는 다음과 같다

1)직렬화/비직렬화 처리

2)인증 체크

3)사용량 제한 체크(호출 허용량 범위인지)

4)권한 클래스지정: 비인증/인증 유저에 대한 해당 API 호출을 허용할 것인지를 결정

5)요청된 API 버전 문자열을 탐지해 request.version에 저장

**여기까지 이뤄지면** method에 대한 함수로 이동

텍스트이(가) 표시된 사진

자동 생성된 설명🡺요렇게 만들기 가능(직접 구현 예시)

-참고로 APIView내부에서 csrf\_exempt장식자가 이미 있기 때문에 POST 요청에서 csrf token을 검사하지 않는다.

-@api\_view는 내가 어떤 작업을 수행할 것인지 미리 List로 작성해 놔야 한다.

Ex) @api\_view([‘GET’,’POST’]) 비어 있으면 매칭이 안된다.

-내부에서는 if문으로 작업을 처리해준다. 따라서 하나의 작업만을 구현하고자 하면 @api\_view를 사용하는 게 편하다.

**<Mixins 상속을 통한 APIView>**

-DRF에서 지원하는 mixins는 CreateModeMixin(create하는 로직이 들어있다), ListModelMixin (리스트 조회하는 로직), RetrieveModelMixin, UpdateModelMixin , DestroyModelMixin

-핵심은 로직만 Mixin함수에 있고 method별로 로직을 연결하는 것이다..

Ex)CreateModelMixin 내부에는 create함수에 serializer가있고 검증, perform\_create, header등의 작업이 내부적으로 구현되어 있다.

Ex)

Class PostListAPIView(mixins.ListModelMixin, mixins.CreateModelMixin**, generic.GenericAPIView**):

여기서 받는 인자의 순서가 중요함 generic.GenericAPIView는 반드시 뒤에 와야 된다. 파이썬의 클래스 상속의 특성 때문(MRO)

그리고 이렇게 만들었을 때 self.list, self.create 즉 minxins의 함수값을 호출해서 return 값을 줄 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명🡺Retrive나 update, Destroy에 따라 .뒤에 붙는 호출 함수가 다르다.

근데 저렇게 매번 호출 하는건 불편하다! 따라서 발전은 **APIView🡺 Mixins🡺 Generic🡺Viewset**으로 발전해 간 것.

텍스트이(가) 표시된 사진

자동 생성된 설명

따라서 **Generic APIVIew**를 활용하는 법을 아는 게 편하다.

<Generic APIView와 그냥 APIView의 코드상 차이점>

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이렇게 generics의 APIView는 매우 간단하게 2줄로 끝낼 수 있지만 Mixins는 매핑을 하나씩 해야 된다.

-이렇게 다양한 View의 구현 방법이 있다.

**<ViewSet과 Router>**

-Viewset은 단일 리소스에서 관련있는 View들을 단일 클래스에서 제공해 준다.

2개의 URL이 있다하자 list/create/detail/update/partial\_upadate/delete 즉 list/create에 대한 URL과 이후 URL이 있으면 2개의 CBV가 필요로 한다.

🡺**2개에대한 CBV를 한 개의 클래스로 지원하는 것이 ViewSet**이다.

텍스트이(가) 표시된 사진

자동 생성된 설명viewset으로 list와 retreieve를 한 번에 구현했다

-근데 이것들을 한 번에 정리해 놓은 Viewset이있으니 바로 **ModelViewSet**이다. ModelViewset은 2개가 있는데

**1)ReadOnlyVeiwSet**(Get요청에만 반응하도록 설계 조회/삭제 전용이다.)과

**2)ModelViewSet(list/create, detail/update/partial\_update/delete 모두 지원)**이다.

-URL Patterns에 매핑하는 방법으로는 2가지가 있다

텍스트이(가) 표시된 사진

자동 생성된 설명

라우터는 기능이 조금 더 추가되어 있음🡺**format 인자를 자동으로 패턴에 추가해준다**(api, json방식으로 볼 수 있게 해준다) 그리고 **api-root도 기본적으로 지원**해줘서 현재 router에 등록된 ViewSet의 내역을 조회할 수 있게 해준다.

**<ViewSet에 새로운 EndPoint 추가>**

@action이라는 장식자문법을 사용할 수 있다.

**-@action(detail=? , methods=[‘GET’])** 이렇게 사용하는데 detail을 통해 viewset에 있는 url중 어떤 url을 사용할 것인가를 결정하고 method는 어떤 method에 대해 반응할지 정할 수 있다

<Renderer를 통한 다양한 응답 포맷 지원>

-Renderer란 같은 Endpoint에서 요청받은 타입에 맞춰서 응답 포맷을 지원해 준다.

-URL 방법이나 Content-Type의 방법을 통해 Renderer을 지정 가능하다.

기본 지원되는 Renderer

1)JSONRenderer: json.dumps를 통한 JSON 직렬화, format은 json 미디어타입은 application/json

2)BrowsableAPIRenderer: self-document HTML 렌더링 format은 api, media\_type는 text/html이다.

3)TemplateHTMLRenderer: 지정 템플릿을 통한 렌더링 format은 api media\_type은 text/html이다, 경우에 따라서 HTML 응답을 받을 수 있다(serverside-rendering)

-**TemplateHTMLRender**는 Template즉 HTML파일로 Rendering을 수행하기 때문에 Seriallizer가 불필요 하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

<Third Party 랜더러>

텍스트이(가) 표시된 사진

자동 생성된 설명이렇게 있으므로 내가 필요한 상황에서의 renderer를 사용하면 된다.

<Renderer 선택>

-**전역 지정 시** 🡪settings의 REST\_FRAMEWORK의 DEFAULT RENDERER CLASSES에 리스트 문자열로 지정한다.

-APIView마다 지정 시 (특정 API View에서만 사용할 경우)🡺queryset, serializer\_class와 더불어 renderer\_classes 리스트 작성

-@api\_view 마다 지정🡪renerer\_classes 장식자!

어지간~ 하면 전역지정이 편하다.

<**Response**의 기본 2가지 유형의 응답 포맷>

1)api: API Endpoint에 브라우저를 통해 접근할 때, 웹 UI로 조회 가능

2)json: 보통의 API 접근.

<응답 포맷 결정>

-Accept 헤더는 Accept: application/json, text/html이 있다

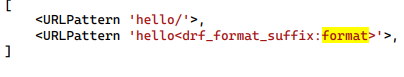
-**함수 기반 뷰**에서는 필히 urlpatterns에 지정된 인자를 함수에서 키워드 인자로 받아야 한다!

-format이 들어오지 않을 수 있으므로 format=None은 기본적으로 넣어 둔다

-format\_suffix\_patterns를 import하면 패턴리스트를 통해 패턴을 추가할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명즉 이렇게 된다면 **hello.<format>**이 되는 거다. (패턴 뻥튀기!)



**<Form과 Serializer 관점에서 DRF 비교>**

-Serializer는 Serializer와 ModelSerializer가 있음 이는 데이터 변환/직렬화 지원을 함! ModelSerializer는 Serializer 생성을 위한 shortcut이다. 🡺QuerySey/Model 객체를 Native Python 데이터 타입과 JSON/XML등으로 바꿔준다.

-장고의 Form과 ModelForm과 Serializer는 유사하다.

**-기능상 비교**

1)Form/ModelForm 🡺**HTML 입력form을 통한 입력에 대한 유효성 검사**, Create/Update에 대한 처리에서 활용하며 **장고 admin에서 활용된다**. CreateView/UpdateView CBV를 통한 뷰처리

2)Serailzer/ModelSerializer🡺 데이터 변환 및 직렬화 지원(JSON 포멧), 주로 JSON 포맷 입력에 대한 유효성 검사를 한다. List/Create 및 특정 Record에 대한 Retrieve/Edit/Delete에서 활용한다

APIView를 통한 뷰 처리를 한다(단일 View), ViewSet으로 2개뷰와 2개의 URL 처리가능(Router?)

**-주된 호출 주체**

1)Form: 일반적으로 웹 브라우저 상에서 HTML From Submit, JavaScript에 의한 비동기 호출이 된다. +안드로이드 ios도 응답 가능  
2)Serializer: 다양한 **Client**에 대한 Data 위주의 Http(s)요청

<코드상으로 비교해보기!>

-클래스 정의 시

텍스트이(가) 표시된 사진

자동 생성된 설명

🡺여기서 Form은 Form.save를 지원해 준다. 보면 구조상으로는 거의 비슷하다고 볼 수 있다. 하지만 widget같은 것은 Serializer에 없다.

-FBV를 통한 요청/응답 시

텍스트이(가) 표시된 사진

자동 생성된 설명

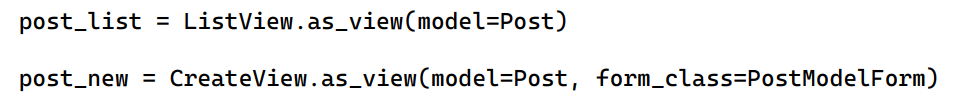
텍스트이(가) 표시된 사진

자동 생성된 설명

🡺차이점은 우선 Form은 PostForm(request.POST, request.FILES)를 받고 Serializer는 data=request.POST를 받는다는점이고 .is\_valid()는 같다. 그리고 응답시 redirect(또는 JsonResponse)와 Response의 차이점이 있다.

-CBV를 통한 요청 및 응답

먼저 장고 기본에서는 ListView와 CreateView를(from **generic**) 따로 만들어 주고 매핑해주면 된다.



텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 DRF에서는 APIVIew를 활용해 ListCreateAPIVIew를 만들어서 매핑 시켜주면된다.

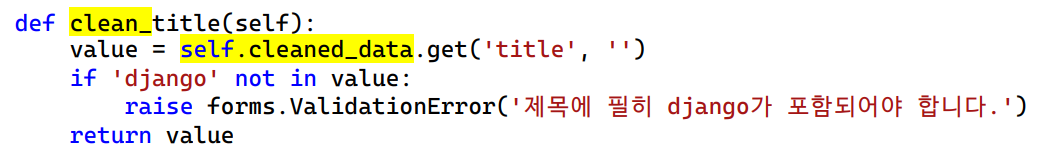
텍스트이(가) 표시된 사진

자동 생성된 설명(단일 URL에서 list/create 요청 처리가 중요!)

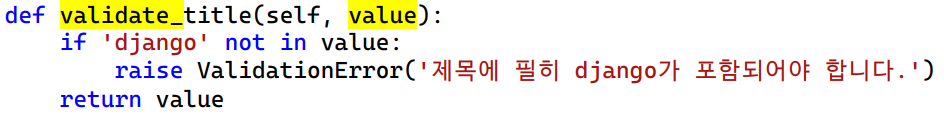
-유효성 검사는 Serializer는 인스턴스를 만들어서(get\_serailizer(data=request.data)를 통해서) 인스턴스의 유효성을 검사한다. if문을 쓰지않고 is\_valid(raise\_exception=True)를 사용한다.

**-커스텀 유효성 검사 루틴**

1)장고 기본에서는 clean\_\* 이라는 함수명이 커스텀 유효성 검사를 수행한다. 데이터는 cleanded\_data.get을 통해 가져온다.



2)DRF에서는 validate\_\* 라 붙은 함수에서 커스텀 유효성 검사가 발생!



**<Serializer를 통한 유효성 검사 및 저장>**

-Serializer의 생성자는 Django의 Form과 컨셉/사용법이 유사하나 생성자의 차이가 있다!

-**data**인자가 주어지면 **is\_vaild()**가 호출되고 나서야 initial\_data필드에 접근 가능하고 유효성 검증이 가능하다. 통과한 값들은 .save()시에 사용된다! .errors는 유효성 검증 수행 후에 오류내역을 보여주고 .data는 유효성 검증 후에 갱신된 인스턴스에 대한 필드 값 사전이다!

-데이터 직렬화를 위해서는 instance인자에 모델 객체나 Qs를 넘겨주면 되고 유효성 검사를 위해서는 data에 입력값을 넘겨준다.

-save(\*\*kwarg) 호출 시 DB에 저장한 관련 instance를 리턴하고 validated\_data와 kwarg 사전을 합친 데이터를 **실제 DB에 업데이트하려고** 시도한다.

-DRF에서는 **유일성 체크**를 도와주는 Vaildators를 제공해 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-UniqueVaildator는 모델 Field에 unique=True로 지정하면 자동 지정, queryset은 필수로 필요하고 message, lookup(디폴트 exact)을 지정 가능하다.

모델 필드에 지정해도 되고 Serializer 필드에 직접 validators 지정 가능

텍스트이(가) 표시된 사진

자동 생성된 설명

-UniqueForDateValidator(Date에 Month/Year 넣기 가능) 🡺 queryset과 field와 date\_field가 필수 인자값이고 message를 작성 할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명🡺특정 범위의 날짜에서의 필드가 유니크한지 검사한다!

-유효성 검사 실패시 VaildationError 예외가 발생한다. 반드시 rest\_framework.exceptions. ValidationError를 사용할 것 장고 기본에서는 Django.forms.exceptions. VaildationError를 사용하기 때문에 이름이 같다!

-Serializer에서의 유효성 검사 🡺필드 정의시에 Validators를 지정하거나 클래스 Meta.validators를 지정해야 된다.

++ Field Level에서의 검사🡺 def validate\_title로 지정, Object Level의 검사🡺 def validate로 검사!

-DB로의 반영과 Mixins의 **perform\_** 계열 함수

이렇게 perform\_ 계열의 함수를 통해 **실질적인 DB 처리 로직이 이뤄진다**.

**<Authentication과 Permission>**

-Authentication은 인증으로 유입되는 요청을 허용/거부하는 것을 결정하는 게 아닌 **단순 인증 정보로 유저를 식별하는 것**이다.

-인증🡪Permissions(각 요청에 대한 허용/거부)🡪Throttling(일정 기간동안 허용할 최대 요청 수) 로 인증과 허가가 진행된다.

**-인증 처리 순서**

1)매 요청시마다 APIView의 dispatch(request) 호출 (클래스 기반 View일 경우!)

2)APIVIew의 initial(request) 호출

3)APIView의 perform\_authentication(request) 호출, 실제 API에서는 토큰을 사용해 인증 가능하고 or Username같은 것을 계속 보내는 방식이 있다.

4)request의 user 속성 호출

5)request의 \_authenticate() 호출, 실제적으로 인증을 확인한다. 실패 시 not\_authenticate함수 호출

-지원하는 인증 종류

1)**SessionAuthentication**: 세션을 통한 인증, APIView의 디폴트 값. 한 번만 로그인 하면 세션에 로그인 정보가 저장된다.

2)**BasicAuthentication**: basic **인증 헤더**를 통한 인증, 헤더에 유저명+ 암호가 들어가 있다. 그래서 항상HTTPS 인증서를 통해 보내야 한다 위협에 노출 가능!

3)**TokenAuthentication**: 토큰 헤더를 통해 인증한다. 토큰은 unique한 토큰이다 즉 유출되면 X

만료기간도 없기 때문에 유출되면 안된다.

4)**RemoteAuthentication**: User가 다른 서비스에서 관리될 때 Remote인증

-인증 허가를 할 때 개체에 대한 접근은 **인증/식별 만으로는 충분하지 않다**. 추가로 **각 개체에 대한 허가(Permission?)이 필요**로 하다.

-DRF의 Permission 시스템: 현재 요청에 대한 허용/거부 결정, **APIView 단위로 지정 가능**!

참고로 장고와 DRF의 Permission은 다르다!

1)**AllowAny**: 디폴트 전역 설정, 인증 여부관계없이 뷰 호출을 허용한다

2)**IsAuthencticated**: 인증된 요청에 한해서 뷰 호출 허용

3)**IsAdminUser**: **Staff 인증 요청**에 한해서 뷰 호출 허용(is\_active, is\_staff, is\_superuser등이 있는데 여기서 is\_staff를 체크한다.)

4)**IsAuthenticatedOrReadOnly**: 비 인증 요청에는 읽기 권한만 허용

5)**DjangoModelPermissions**: 인증된 요청에 대해 뷰 호출 허용+장고 모델단위 Permission 체크

6) **DjangoModelPermissionsOrAnonReadOnly**: 5에서 비 인증 요청 시 읽기전용 허용

7)**DjangoObjectPermissions**: 비인증 요청은 거부하고 인증된 요청은 Object에 대한 권한 체크

-APIView와 @api\_view일 경우 차이점 (CBV와 FBV일 때 permission 설정법?)

1)**APIView**는 permission\_classes=[Permission 시스템중 하나]로 설정 하고

2**)@api\_view**는 @permission\_classes([])를 사용하면 된다.

-디폴트 전역 설정은 **AllowAny**임을 기억!

<커스텀 Permission>

-모든 Permission 클래스는 다음 2가지 함수를 선택적으로 구현함

1)**has\_permission(request, view)** 🡺APIView 접근 시 체크한다, 대부분의 Permission 클래스에서 구현하면 로직에 따라 True/False 반환

2)**has\_object\_permission(request ,view, obj)** 🡺APIView의 get\_object 함수를 통해 object 획득 시 체크, 브라우저를 통한 API 접근에서 Create/Update Form 노출 시 체크, DjangoObjectPermissions 에서 구현하며 로직에 따라 T/F 반환

**<Filtering과 Ordering>**

-필터링의 인자는 APIView와 @장식자사이의 차이점이 존재

테이블이(가) 표시된 사진

자동 생성된 설명

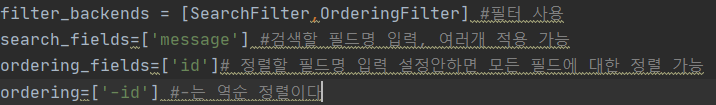
-get\_queryset이란 함수를 사용해 요청마다 필터링 가능하다.

-기본 장고를 사용했을 경우에는 @admin\_register를 사용해 admin 페이지에 등록했음

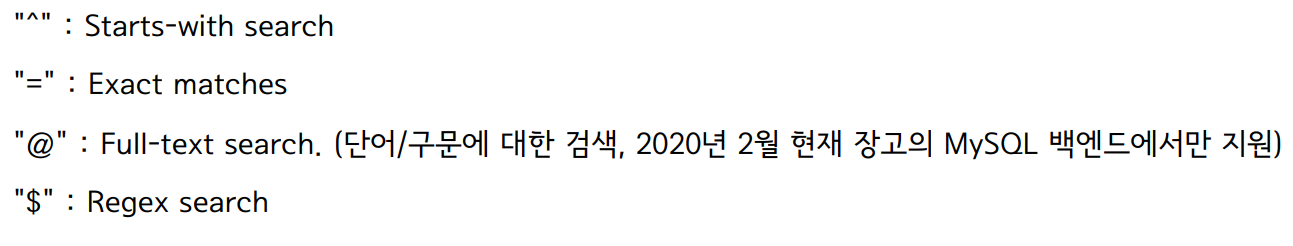
텍스트이(가) 표시된 사진

자동 생성된 설명

-DRF에서는 지원해주는 필터를 사용할 수 있다. 예시로 SearchFilter와 OrderingFilter를 APIView에서 사용하면 Admin에서 설정한 값을 설정 가능하게 된다.



-Search\_fields의 문자열 패턴 지정을 통해 타이트하게 범위를 지정할 수 있다. 종류는 4가지다



^는 ^메시지로 시작하는 것만 찾고 =는 포함된 것들만 나온다. 일반적으로는 잘 사용 X

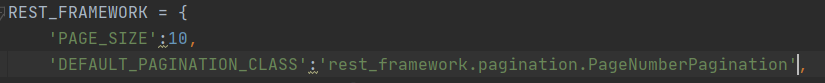
**<Pagination>**

-DRF에사 기본 지원하는 페이징 방식은 2가지

1)**PageNumberPagination**: page/page\_size 인자를 통한 페이징 처리

2)**LimitOffsetPagination**: offset/limt인자를 통한 페이징 처리

-page\_size의 경우는 미지정 상황을 위해 Default값을 지정하는 게 좋다 🡺settings.py의 REST\_FRAMEWORK ={“PAGE\_SIZE”:10} 이렇게 지정 +Limitoffset도 비슷하게 설정 가능하다.



🡺이렇게 settings.py에 지정해 두면 편하다!

-pagination을 설정하면 Post List 구조가 바뀌게 된다. Result란 값 안에 json 객체가 들어있다.

**<Throttling 🡺 최대 호출 횟수 제한하기>**

-용어 정리

1)Rate: 지정 기간 내에 허용할 최대 호출 횟수

2)Scope: 각 Rate에 대한 별칭(alias)

3)Throttle: 특정 조건 하에 최대 호출 횟수를 결정하는 로직이 구현된 클래스

-기본 제공되는 Throttle

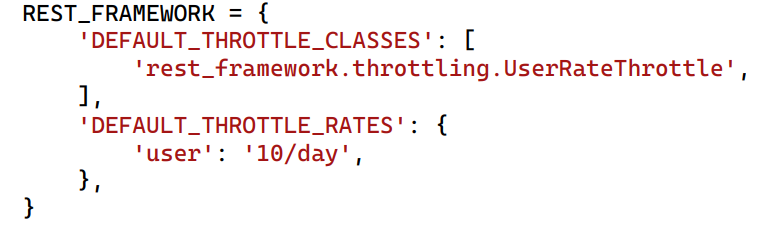
1)**AnonRateThrottle**: 인증 요청에 제한이 x, 비 인증 요청에는 IP 단위로 횟수 제한, 디폴트 scope는 ‘anon’

2)**UserRateThrottle**: 인증 요청 시 유저 단위로 횟수 제한, 비 인증 요청 시 IP단위로 횟수 제한. 디폴트 scope:’user’이다

3)**ScopedRateThrottle**: 2)랑 같지만 APIView내 throttle\_scope 설정을 **읽어 APIView별로 별도의 scope 적용 가능**

-디폴트 설정 시 Throttle은 설정되어 있지 않다! (settings.py에서는) 따라서 settings.py의 내용을 수정하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명🡺 이렇게 설정하면 하루에 user 10명 제한을 걸어둔다는 의미이다. 비인증 요청에 대한 거부는 Permission으로 설정하면 된다.

-최대 호출 횟수 넘을시 429 ‘Too Many Request”에러가 발생

<Cache>

-장고에서 **매 요청시 마다 cache에서 timestamp list를 get/set해준다**. 🡺캐시 성능이 중요하다

-SimpleRateThrottle에서는 다음과 같이 Default 캐시를 설정해 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-장고에서는 다양한 캐시 지원을 하는데 주로 **Memcached**와 **Redis**를 많이 사용한다. 디폴트는 **로컬 메모리 캐시**이다.

-로컬 메모리 캐시는 서버를 껏다 키면 데이터가 날라가서 쓰로틀링도 초기화 된다.

<설정>

-Rate 포맷은 **“숫자/간격**”으로 설정한다 숫자🡺최대 요청 제한 횟수, 간격🡺지정 문자열의 첫글자만 사용(d, s, m ,h 등등)

Rates 제한 메커니즘에서 SingleRateThrottle에서는 요청한 시간의 timestamp를 list로 유지한다 즉 매 요청마다 다음과 같은 작업을 수행

1)cache에서 **timestamp list를 가져오기**🡪

2)체크 범위 값 밖의 timestamp 버리기 🡪

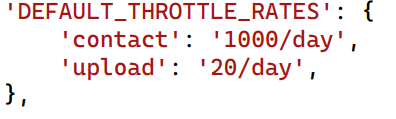
3) timestamp list의 크기가 허용 범위보다 크면 요청 거부 (429)🡪

4) 작을경우 timestamp를 list에 추가 cache에 다시저장

-클라이언트 IP 🡺**X-Forward-For** 헤더와 **REMOTE\_ADDR** 헤더를 참조해서 확정 X-Forward-For가 우선순위가 더 높다.

-API 별로 서로 다른 Rate 적용하기 위해서는 DEFAULT\_THROTTLE\_RATES에 **별칭을 만들어 scope로 참조하면된다**.

Ex) ‘contact’:’1000/day’, ‘upload’:’20/day’ 이렇게 만들어 주고 scope=’contact’, scpoe=’upload’ 이렇게 API 별로 다른 Rate를 적용하면 된다.

이렇게 만들어 주고

텍스트이(가) 표시된 사진

자동 생성된 설명🡺throttle.py에서 scope 설정 후

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명🡺views.py에서 클래스 설정을 해준다.

이 방법 말고 views.py에 직접! Throttle\_scope=’contact’로 설정해 주면된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-만약 Rate를 커스터 마이징 하고 싶다면 **allow\_request 함수를 설정해 줘라**!