



Django5

💻 강의날짜	@2022/10/17
⌚ 작성일시	@2022년 10월 17일 오후 12:22
⌚ 편집일시	@2022년 10월 17일 오후 6:50
🏷️ 분야	django
🏷️ 공부유형	강의
☑️ 복습	□
☰ 태그	Django REST REST API Response JSON

REST API

HTTP

- HyperText Transfer Protocol
- HTML 문서와 같은 리소스들을 가져올 수 있도록 하는 프로토콜(규칙, 약속)
- 클라이언트와 서버는 다음과 같은 개별적인 메시지 교환에 의해 통신
 - 요청(request)
 - 클라이언트에 의해 전송되는 메시지
 - 응답(response)
 - 서버에서 응답으로 전송되는 메시지

HTTP 특징

- **Stateless (무상태)**
 - 동일한 연결(connection)에서 연속적으로 수행되는 두 요청 사이에 링크가 없음

- 즉, 응답을 마치고 연결을 끊는 순간 클라이언트와 서버 간의 통신이 끝나며 상태 정보가 유지되지 않음

HTTP Request Methods

- 리소스에 대한 행위(수행하고자 하는 동작)을 정의
- 즉, 리소스에 대해 수행할 원하는 작업을 나타내는 메서드 모음을 정의
- HTTP verbs 라고도 함
- HTTP Method 예시
 - `GET`, `POST`, `PUT`, `DELETE` ...

대표 HTTP Request Methods

1. `GET`
 - a. 서버에 리소스의 표현을 요청
 - b. `GET`을 사용하는 요청은 데이터만 검색해야 함
2. `POST`
 - a. 데이터를 지정된 리소스에 제출
 - b. 서버의 상태를 변경
3. `PUT`
 - a. 요청한 주소의 리소스를 수정
4. `DELETE`
 - a. 지정된 리소스를 삭제

HTTP response status codes

- 특정 HTTP 요청이 성공적으로 완료되었는지 여부를 나타냄
- 응답은 5개의 그룹으로 나뉨
 1. Informational responses (100-199)
 2. Successful responses (200-299)
 3. Redirection messages (300-399)
 4. Client error responses (400-499)
 5. Server error responses (500-599)

Identifying resources on the Web

개요

- 웹에서 리소스를 식별하는 방법에 대해 학습

웹에서의 리소스 식별

- HTTP 요청의 대상을 리소스(resource, 자원)라고 함
- 리소스는 문서, 사진 또는 기타 어떤 것이든 될 수 있음
- 각 리소스는 식별을 위해 **URI**로 식별됨

URI

- Uniform Resource Identifier (통합 자원 식별자)
- 인터넷에서 하나의 리소스를 가리키는 문자열
- 가장 일반적인 URI는 웹 주소로 알려진 **URL**

URL 예시

```
https://docs.djangoproject.com  
https://docs.djangoproject.com/en/4.1/intro/  
https://docs.djangoproject.com/en/4.1/search/?q=model
```

- 특정 이름공간에서 이름으로 리소스를 식별하는 URI는 **URN**

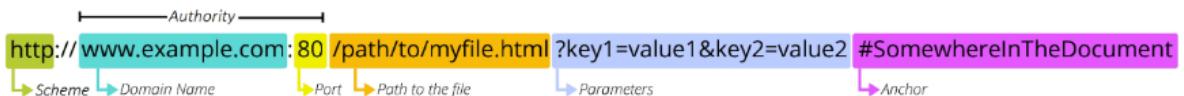
URN 예시

```
# ISBN(국제 표준 도서번호)에서 식별되는 "로미오와 줄리엣" 도서  
urn:isbn:9788937461736  
  
# ISAN(국제 표준 시청각 자료번호)에서 식별되는 "2002년작 영화 스파이더맨"  
urn:isbn:0000-0000-2CEA-0000-1-0000-0000-Y
```

URL

- Uniform Resource Locator (통합 자원 위치)
- 웹에서 주어진 리소스의 주소
- 네트워크 상에 리소스가 어디 있는지(주소)를 알려주기 위한 약속

- 이러한 리소스는 HTML, CSS, 이미지 등이 될 수 있음
- URL은 다음과 같이 여러부분으로 구성되며 일부는 필수이고 나머지는 선택사항



URL 구조

- **Scheme (or protocol)**
 - 브라우저가 리소스를 요청하는데 사용해야 하는 프로토콜
 - URL의 첫 부분은 브라우저가 어떤 규약을 사용하는지를 나타냄
 - 기본적으로 웹은 HTTP(S)를 요구하며 메일을 열기 위한 `mailto:`, 파일을 전송하기 위한 `ftp:` 등 다른 프로토콜도 존재

`https://www.example.com:80/path/to/myfile.html/?key=value#quick-start`

19

- **Authority**
 - Scheme 다음은 문자 패턴 `://` 으로 구분된 Authority(권한)이 작성됨
 - Authority는 domain과 port를 모두 포함하며 둘은 `:`(콜론)으로 구분됨

`https://www.example.com:80/path/to/myfile.html/?key=value#quick-start`

20

1. Domain Name
 - a. 요청 중인 웹 서버를 나타냄
 - b. 어떤 웹 서버가 요구되는지를 가리키며 직접 IP주소를 사용하는 것도 가능
 - c. 하지만, 사람이 외우기 어렵기 때문에 주로 Domain Name으로 사용
 - d. 예를 들어 도메인 `google.com` 의 IP주소는 `142.251.42.142`

`https://www.example.com:80/path/to/myfile.html/?key=value#quick-start`

21

2. Port
 - a. 웹 서버의 리소스에 접근하는데 사용되는 기술적인 문 (Gate)

- b. HTTP 프로토콜의 표준 포트는 다음과 같고 생략이 가능 (나머지는 생략 불가능)
 - i. HTTP - 80
 - ii. HTTPS - 443
- c. Django의 경우 8000(80+00)이 기본 포트로 설정되어 있음

<https://www.example.com:80/path/to/myfile.html/?key=value#quick-start>

- **Path**

- 웹 서버의 리소스 경로
- 초기에는 실제 파일이 위치한 물리적 위치를 나타냈지만, 오늘날은 실제 위치가 아닌 추상화된 형태의 구조를 표현
- 예를 들어 `/articles/create/` 가 실제 articles 폴더 안에 create 폴더 안을 나타내는 것은 아님

<https://www.example.com:80/path/to/myfile.html/?key=value#quick-start>

- **Parameters**

- 웹 서버에 제공하는 추가적인 데이터
- 파라미터는 `'&'` 기호로 구분되는 key-value 쌍 목록
- 서버는 리소스를 응답하기 전에 이러한 파라미터를 사용하여 추가 작업을 수행할 수 있음

<https://www.example.com:80/path/to/myfile.html/?key=value#quick-start>

- **Anchor**

- 리소스의 다른 부분에 대한 앵커
- 리소스 내부 일종의 **북마크**를 나타내며 브라우저에 해당 북마크 지점에 있는 콘텐츠를 표시
 - 예를 들어 HTML 문서에서 브라우저는 앵커가 정의한 지점으로 스크롤 함
- fragment identifier(부분 식별자)라고 부르는 `'#'` 이후 부분은 서버에 전송되지 않음

- 예를 들어 <https://docs.djangoproject.com/en/3.2/intro/install/#quick-install-guide> 요청에서 `#quick-install-guide` 는 서버에 전달되지 않고 브라우저에게 해당 지점으로 이동할 수 있도록 함

<https://www.example.com:80/path/to/myfile.html?key=value#quick-start>

정리

- 앱에서의 리소스 식별
 - 자원의 식별자 (URI)
 - 자원의 위치로 자원을 식별 (URL)
 - 고유한 이름으로 자원을 식별 (URN)

REST API

API

- Application Programming Interface
- 애플리케이션과 프로그래밍으로 소통하는 방법
 - 개발자가 복잡한 기능을 보다 쉽게 만들 수 있도록 프로그래밍 언어로 제공되는 구성
- API를 제공하는 애플리케이션과 다른 소프트웨어 및 하드웨어 등의 것들 사이의 간단한 계약(인터페이스)이라고 볼 수 있음
- API는 복잡한 코드를 추상화하여 대신 사용할 수 있는 몇 가지 더 쉬운 구문을 제공
 - 예를 들어 집의 가전 제품에 전기를 공급해야 한다고 가정해보자
 - 우리는 그저 가전 제품의 플러그를 소켓에 꽂으면 제품이 작동함
 - 중요한 것은 우리가 가전 제품에 전기를 공급하기 위해 **직접 배선을 하지 않는다는 것**
 - 이는 매우 위험하면서도 비효율적인 일

WEB API

- 웹 서버 또는 웹 브라우저를 위한 API
- 현재 웹 개발은 모든 것을 하나부터 열까지 직접 개발하기보다 여러 API를 활용하는 추세

- 대표적인 Third Party Open API 서비스 목록
 - Youtube API
 - Naver Papago API
 - Kakao Map API
- API는 다양한 타입의 데이터를 응답
 - `HTML` , `XML` , `JSON` 등

REST

- Representational State Transfer
- API Server를 개발하기 위한 일종의 소프트웨어 설계 방법론
 - 2000년 로이 필딩의 박사학위 논문에서 처음으로 소개 된 후 네트워킹 문화에 널리 퍼짐
- **소프트웨어 아키텍쳐 디자인 제약 모음**
 - (a group of software architecture design constraints)
- REST 원리를 따르는 시스템을 **RESTful** 하다고 부름
- REST의 기본 아이디어는 리소스 즉 자원
 - **자원을 정의하고 자원에 대한 주소를 지정하는 전반적인 방법을 서술**

REST에서 자원을 정의하고 주소를 지정하는 방법

1. 자원의 식별
 - a. URI
2. 자원의 행위
 - a. HTTP Method
3. 자원의 표현
 - a. 자원과 행위를 통해 궁극적으로 표현되는 (추상화된) 결과물
 - b. JSON으로 표현된 데이터를 제공

JSON

- JSON is a lightweight data-interchange format
- JavaScript의 표기법을 따른 단순 문자열

- 파이썬의 dictionary, 자바스크립트의 object처럼 C계열의 언어가 갖고 있는 자료구조로 쉽게 변환할 수 있는 **key-value 형태의 구조**를 갖고 있음
- 사람이 읽고 쓰기 쉽고 기계가 파싱(해설 & 분석)하고 만들어내기 쉽기 때문에 현재 API에서 가장 많이 사용하는 데이터 타입

```
{
  "squadName": "Super hero squad",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    }
  ]
}
```

REST 정리

- 자원을 정의하고 자원에 대한 주소를 지정하는 방법의 모음
 - 자원을 식별 - [URI](#)
 - 자원에 대한 행위 - [HTTP Methods](#)
 - 자원을 표현 - [JSON](#)
- 설계 방법론은 지키지 않았을 때 잃는 것보다 지켰을 때 얻는 것이 훨씬 많음
 - 단, 설계 방법론을 지키지 않더라도 동작 여부에 큰 영향을 미치지는 않음
 - 말 그대로 방법론일 뿐이며 규칙이나 규약은 아님

Response JSON

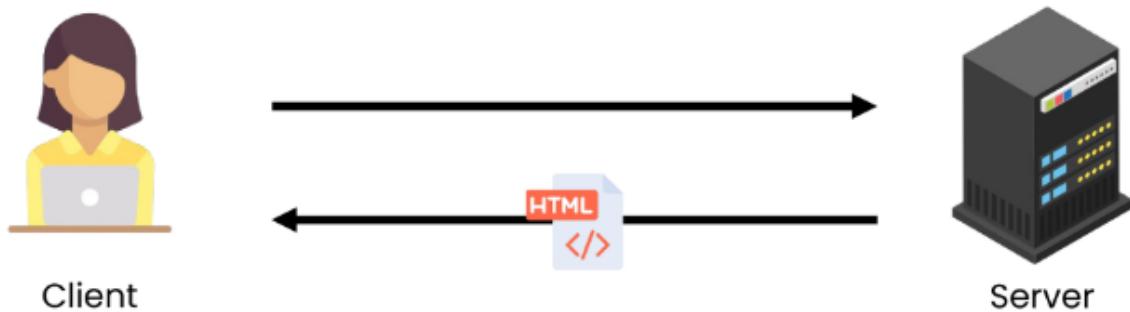
개요

- JSON 형태로의 서버 응답 변화
- 다양한 방법의 JSON 응답

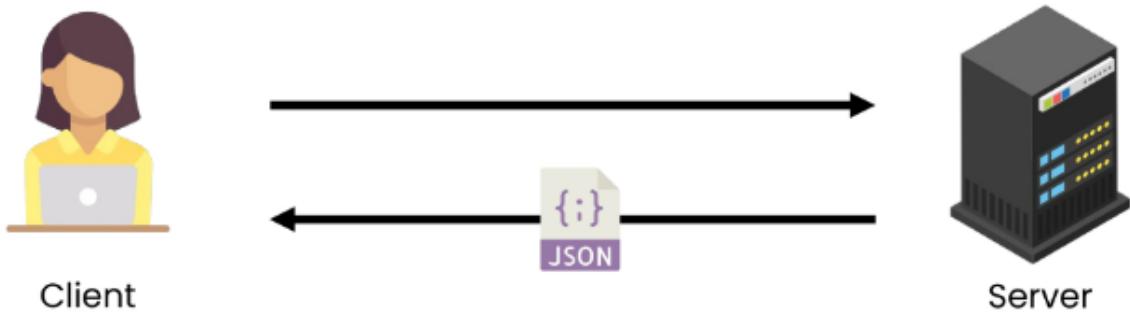
INTRO

서버가 응답하는 것

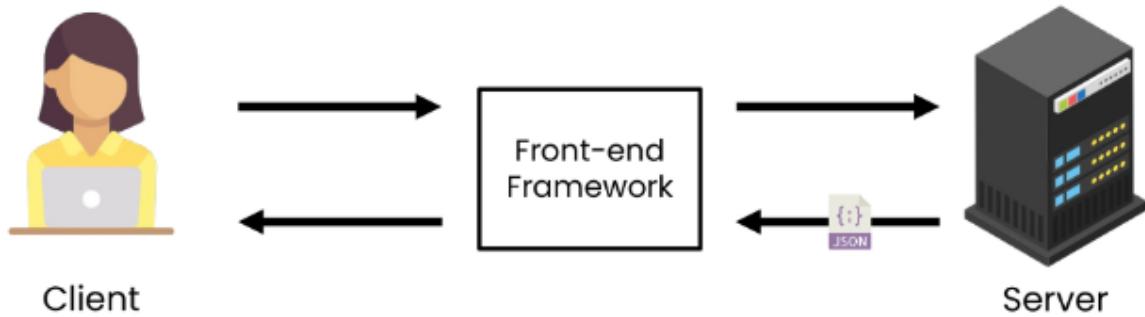
- 지금까지 Django로 작성한 서버는 사용자에게 페이지(html)만 응답하고 있었음
- 하지만 사실 서버가 응답할 수 있는 것은 페이지 뿐만 아니라 다양한 데이터 타입을 응답할 수 있음



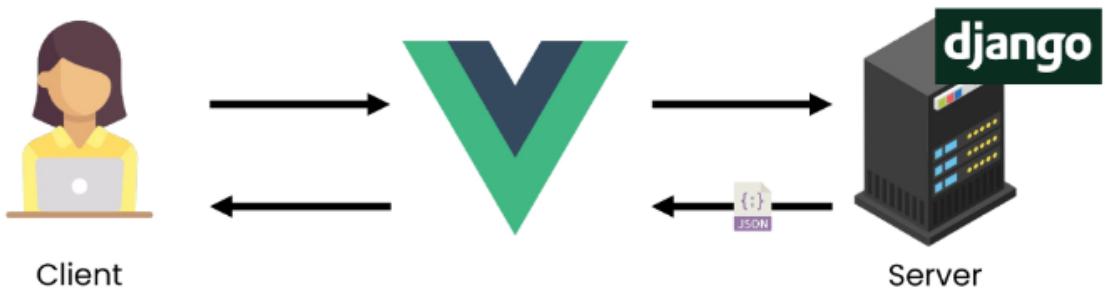
- 페이지(html)를 응답하는 서버



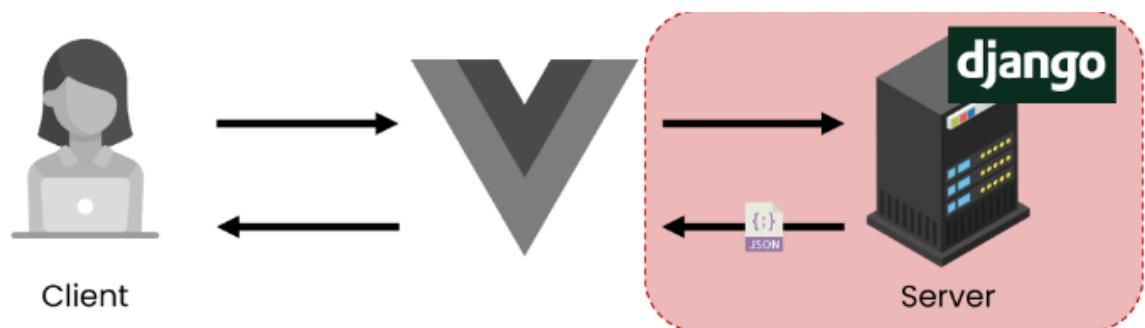
- 이제는 JSON 데이터를 응답하는 서버로의 변환
- 그렇다면 사용자에게 보여질 화면은 누가 구성하게 될까?



- JSON 데이터를 받아 화면을 구성하여 사용자에게 보여주는 것은 Front-end Framework가 담당할 예정



- Front-end Framework는 Vue.js를 사용
- Django는 더 이상 Template부분에 대한 역할을 담당하지 않게 되며 Front-end와 Back-end가 분리되어 구성되게 됨



- 이번 시간에는 JSON을 응답하는 Django 서버를 구성하는 법을 학습

사전 준비

1. 사전 제공된 `01_json_response` 프로젝트 준비
2. 가상 환경 생성, 활성화 및 패키지 설치

3. migrate 진행

```
$ python manage.py migrate
```

4. 준비된 fixtures 파일을 load 하여 실습용 초기 데이터 입력

```
$ python manage.py loaddata articles.json
```

- 입력된 데이터 확인

SQL ▾		
		content
id	title	
1	Land probably you.	Bill third easy employee outside. Cup international bring quality. Relate success degree. Indeed choose actually threat quite partner friend. Important sort bring start space blood pre
2	Economic probably must.	Speak final west. Forget manage lawyer to story sense. Hotel natural blue soldier ready record either home. Information particularly either administration.
3	Sister wind page hour both some.	Class might so reflect away successful. Cell do industry physical possible each. Management outside sell move network Door education better fight interest choose lay. Me huge me upon across.
4	Style strategy school gas two investment college.	Be artist sense. Course film tough remain loss risk begin. Direction large matter such center. Machine purpose career house personal. Land ok represent than. Between weight idea stuff little.
5	Make although environmental during upon beat.	Interview according weight financial into agreement phone. Art seven court authority. Particular real wear anything dr
6	Effectiveness service brother down still enough	Speech that note play. Protection against the possibility of a sudden breakdown of the system. Effectiveness of the system.

- 미리 작성된 프로젝트 둘러보기
- view함수는 필요한 부분만 작성해 나갈 예정

```
# my_api/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('articles.urls')),
]
```

```
# articles/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('html/', views.article_html),
    path('json-1/', views.article_json_1),
    path('json-2/', views.article_json_2),
    path('json-3/', views.article_json_3),
]
```

RESPONSE

개요

- 다양한 방법으로 JSON 데이터 응답해보기

1. HTML 응답

2. `JsonResponse()` 를 사용한 JSON 응답
 3. Django Serializer를 사용한 JSON 응답
 4. Django REST framework를 사용한 JSON 응답
1. HTML 응답
- a. 문서(HTML) 한 장을 응답하는 서버 확인하기
 - b. 지금까지 Django로 응답 해오던 방식

```
# articles/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('html/', views.article_html),
]

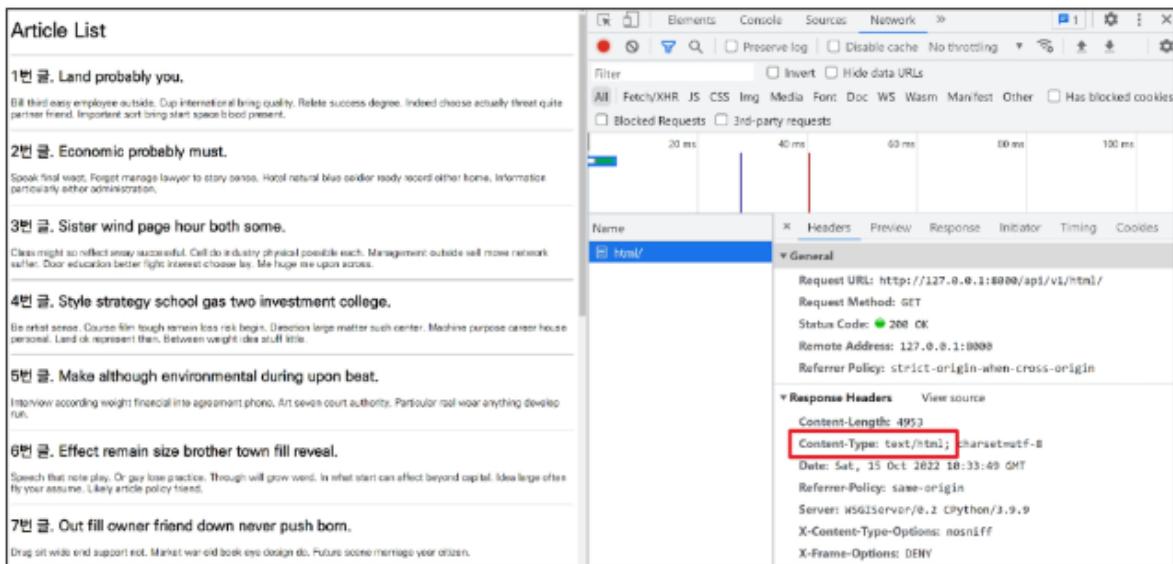
# articles/views.py
from django.shortcuts import render
from .models import Article

def article_html(request):
    articles = Article.objects.all()
    context = {
        'articles': articles,
    }
    return render(request, 'articles/article.html',
    context)
```

```
<!-- articles/article.html -->

<!DOCTYPE html>
<html lang="en">
<head>
    ...
</head>
<body>
    <h1>Article List</h1>
    <hr>
    <p>
        {% for article in articles %}
            <h2>{{ article.pk }}번 글. {{ article.title }}</h2>
            <p>{{ article.content }}</p>
            <hr>
        {% endfor %}
    </p>
</body>
</html>
```

- 응답 페이지 확인



2. JsonResponse() 를 사용한 JSON 응답

- 아제는 문서(HTML) 한 장을 응답하는 것이 아닌 JSON 데이터를 응답해보기
- Django가 기본적으로 제공하는 JsonResponse 객체를 활용하여 Python 데이터 타입을 손쉽게 JSON으로 변환하여 응답 가능

```
# articles/views.py

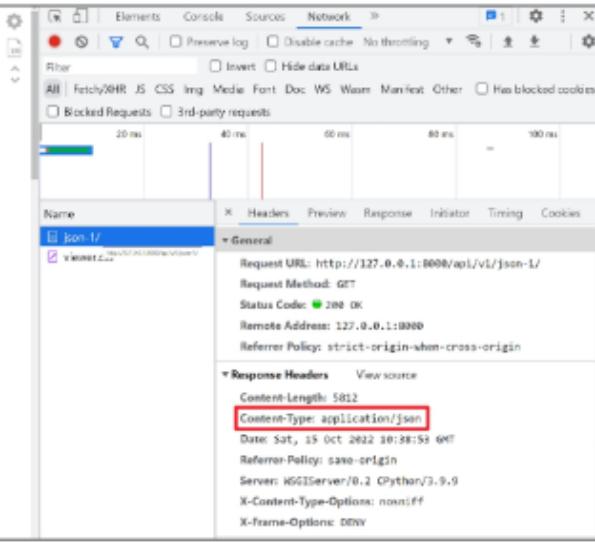
from django.http.response import JsonResponse

def article_json_1(request):
    articles = Article.objects.all()
    articles_json = []

    for article in articles:
        articles_json.append(
            {
                'id': article.pk,
                'title': article.title,
                'content': article.content,
                'created_at': article.created_at,
                'updated_at': article.updated_at,
            }
        )
    return JsonResponse(articles_json, safe=False)
```

- 응답확인

<http://127.0.0.1:8000/api/v1/json-1/>



The screenshot shows the Chrome Network tab with a single request labeled 'bon_1'. The 'Response' tab is selected, displaying the JSON data. The JSON structure is as follows:

```
1 // 2821191523365
2 // http://127.0.0.1:8000/api/v1/json-1/
3
4 [
5   {
6     "id": 1,
7     "title": "Last probably you.",
8     "content": "Will third easy employee outside. Cup international bring quality. Variable success degree. Indeed choices actually threat quite partner friend. Reported sort bring short space blood present.",
9     "created_at": "2015-01-23T09:35:18Z",
10    "updated_at": "2018-08-05T03:29:46Z"
11  },
12  {
13    "id": 2,
14    "title": "Economic probably must.",
15    "content": "Look final sort. Forget manage longer to story voice. Metal natural blue soldier ready record either home. Information particularly either administration.",
16    "created_at": "2009-04-26T10:44:04Z",
17    "updated_at": "2013-09-16T04:52:34Z"
18  },
19  {
20    "id": 3,
21    "title": "Guitar wind page hour both some.",
22    "content": "Class right to reflect away successful. Call on industry physical possible each. Management visible will none network suffer. Under education better flight interest choose lay. We hugo am upon across.",
23    "created_at": "2003-01-13T15:39:58Z",
24    "updated_at": "1974-06-05T21:45:13Z"
25  },
26  {
27    "id": 4,
28    "title": "Style strategy school get two investment collage.",
29    "content": "An artist assess. Course file tough remains less risk begin. Direction tags outer rock center. Reaching purpose career basic personal. Land or represent them. Between weight idea stuff little.",
30    "created_at": "2004-07-26T09:00:42Z",
31    "updated_at": "2019-08-06T03:31:37Z"
32 }
```

❖ 왼쪽처럼 정렬된 json 출력을 보려면
Chrome 확장프로그램 JSON Viewer가 설치 되어있어야 함

3. Django Serializer를 사용한 JSON 응답

- Django의 내장 `HttpResponse()` 를 활용한 JSON 응답
- 이전에는 JSON의 모든 필드를 하나부터 열까지 작성해야 했지만 이제는 그렇지 않음

```
# articles/views.py

from django.http.response import JsonResponse, HttpResponseRedirect
from django.core import serializers

def article_json_2(request):
    articles = Article.objects.all()
    data = serializers.serialize('json', articles)
    return HttpResponseRedirect(data, content_type='application/json')
```

- 응답확인

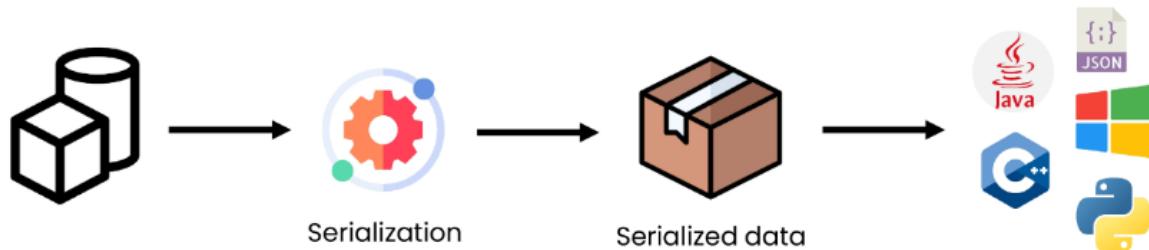
```

1 // 0922103191742
2 // http://127.0.0.1:8000/api/v1/json-2/
3
4 [
5   {
6     "model": "articles.article",
7     "pk": 1,
8     "fields": {
9       "title": "Land probably you.",
10      "content": "Bill third easy employee outside. Cup international bring quality.\nRelate success degree. Indeed choose actually threat quite partner friend. Important sort bring start space blood present.",
11      "created_at": "2015-02-23T05:35:10Z",
12      "updated_at": "2016-08-05T01:26:46Z"
13    }
14  },
15  {
16    "model": "articles.article",
17    "pk": 2,
18    "fields": {
19      "title": "Economic probably must.",
20      "content": "Speak final west. Forget manage lawyer to story sense.\nNote natural blue soldier ready record either home. Information particularly either administration.",
21      "created_at": "2008-01-27T00:44:04Z",
22      "updated_at": "2021-09-10T04:52:34Z"
23    }
24  },
25  {
26    "model": "articles.article",
27    "pk": 3,
28    "fields": {
29      "title": "Sister wind page hour both some.",
30      "content": "Class might so reflect away successful. Cell do industry physical possible each. Management outside sell move network suffer.\nUser education better fight interest choose lay. We hugo me upon across.",
31      "created_at": "2002-09-12T11:29:58Z",
32      "updated_at": "1974-06-05T21:45:13Z"
33    }
34  }
]

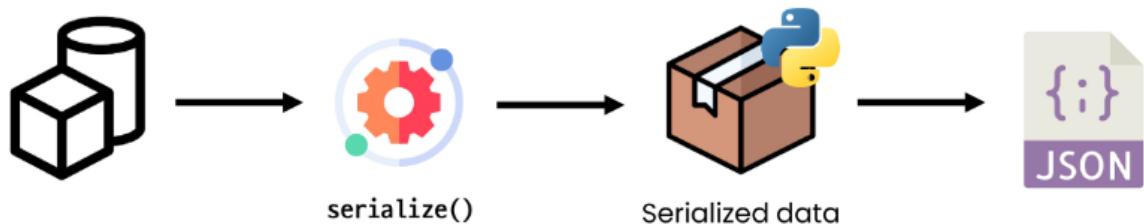
```

Serialization

- 직렬화
- 데이터 구조나 객체 상태를 동일 혹은 다른 컴퓨터 환경에 저장하고, 나중에 재구성할 수 있는 포맷으로 변환하는 과정
 - 즉, 어떠한 언어나 환경에서도 나중에 다시 쉽게 사용할 수 있는 포맷으로 변환하는 과정
- 변환 포맷은 대표적으로 json, xml, yaml이 있으며 json이 가장 보편적으로 쓰임
- 데이터 구조나 객체 상태를 나중에 재구성할 수 있는 포맷으로 변환하는 과정



- Django의 `serialize()` 는 Queryset 및 Model Instance와 같은 복잡한 데이터를 JSON, XML 등의 유형으로 쉽게 변환할 수 있는 Python 데이터 타입으로 만들어 줌



4. Django REST framework를 사용한 JSON 응답

a. Django REST framework (DRF)

- i. Django에서 Restful API 서버를 쉽게 구축할 수 있도록 도와주는 오픈 소스 라이브러리
- ii. Web API 구축을 위한 강력한 toolkit을 제공
- iii. REST framework를 작성하기 위한 여러 기능을 제공
- iv. DRF의 serializer는 Django의 Form 및 ModelForm 클래스와 매우 유사하게 작동

V. <https://www.django-rest-framework.org/>

b. DRF가 설치되어 있는 것을 확인

```
# settings.py

INSTALLED_APPS = [
    'articles',
    'rest_framework',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

- ModelForm과 유사한 ModelSerializer 구조 및 사용법 확인하기

```
# articles/serializers.py

from rest_framework import serializers
from .models import Article

class ArticleSerializer(serializers.ModelSerializer):

    class Meta:
        model = Article
        fields = '__all__'
```

```
# articles/views.py

@api_view(['GET'])
def article_json_3(request):
    articles = Article.objects.all()
    serializer = ArticleSerializer(articles, many=True)
    return Response(serializer.data)
```

- 응답 페이지 확인
- JSON 데이터를 DRF 전용 템플릿으로 응답함

The screenshot shows the Django REST framework's Article Json 3 page. The JSON response is displayed as follows:

```

HTTP/2.0 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept
{
    "id": 1,
    "title": "Django tutorial part 1",
    "content": "In this tutorial we will learn how to build a simple blog using Django framework. This tutorial is for beginners who want to learn how to build a blog from scratch. It covers basic concepts like models, views, templates, urls, and databases. By the end of this tutorial, you will have created a simple blog with multiple posts and comments.",

    "id": 2,
    "title": "Django tutorial part 2",
    "content": "In this tutorial we will learn how to add comments to our blog posts. We will also learn how to handle user authentication using Django's built-in auth system. By the end of this tutorial, you will have a fully functional blog with comments and user authentication.",

    "id": 3,
    "title": "Django tutorial part 3",
    "content": "In this tutorial we will learn how to add users to our blog. We will also learn how to handle user authentication using Django's built-in auth system. By the end of this tutorial, you will have a fully functional blog with users and user authentication.",

    "id": 4,
    "title": "Django tutorial part 4",
    "content": "In this tutorial we will learn how to add users to our blog. We will also learn how to handle user authentication using Django's built-in auth system. By the end of this tutorial, you will have a fully functional blog with users and user authentication.",

    "id": 5,
    "title": "Django tutorial part 5",
    "content": "In this tutorial we will learn how to add users to our blog. We will also learn how to handle user authentication using Django's built-in auth system. By the end of this tutorial, you will have a fully functional blog with users and user authentication."
}

```

On the right, the Network tab of the browser developer tools shows a successful GET request to the endpoint, with a Content-Type header of 'text/html; charset=utf-8'.

직접 requests 라이브러리를 사용하여 json 응답 받아보기

- requests 라이브러리 설치

```
$ pip install requests
```

- 준비된 `gogo.py` 확인

```
# gogo.py

import requests
from pprint import pprint

response = requests.get('http://127.0.0.1:8000/api/v1/json-3/')
result = response.json()

pprint(result)
# pprint(result[0])
# pprint(result[0].get('title'))
```

- Terminal 화면을 나누어 한쪽은 Django 서버를 켜 둔 채로 `gogo.py` 실행하기

The screenshot shows a Jupyter Notebook environment. The code cell contains Python code to make a GET request to a local API endpoint and print the JSON response. The output cell displays the JSON data, which includes a list of posts with their details like title, content, and location.

```
gogo.py
1 # gogo.py
2
3 import requests
4 from sprint import sprint
5
6
7 response = requests.get("http://127.0.0.1:8000/api/v1/json-3/")
8 result = response.json()
9
10 print(result)
11 # sprint(result[0])
12 # sprint(result[0].get('title'))
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
[{"id": 1, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 2, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 3, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 4, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 5, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 6, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 7, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 8, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 9, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 10, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 11, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 12, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}, {"id": 13, "text": "2022-01-20T10:00:00Z", "title": "HTTP/1.1", "body": "Hello", "location": "127.0.0.1:8000 2022-01-20T10:00:00Z"}]
```

+ python - [] python

+ bash - [] bash

정리

- 우리는 DRF를 활용하여 JSON 데이터를 응답하는 Django 서버를 구축할 것

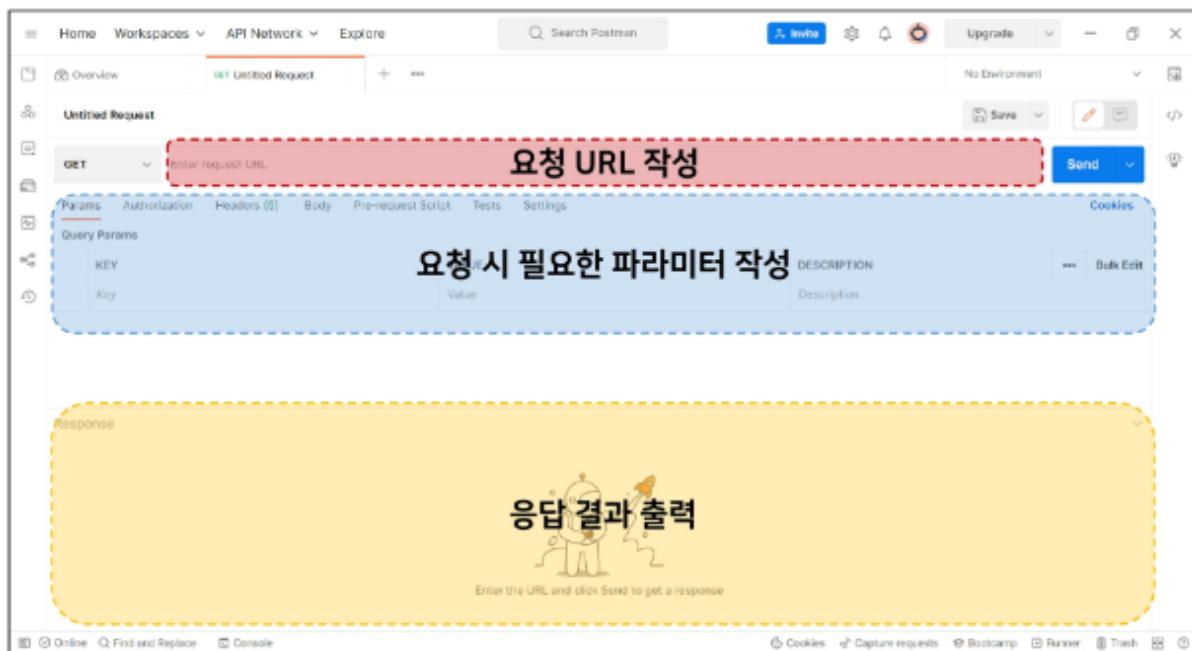
Django REST framework - single Model

개요

- 단일 모델의 data를 Serialization하여 JSON으로 변환하는 방법에 대한 학습

사전 준비

- Postman 설치
 - API를 구축하고 사용하기 위한 플랫폼
 - API를 빠르게 만들 수 있는 여러 도구 및 기능을 제공
 - Postman 화면 구성



1. 준비된 `02_drf` 프로젝트로 진행
2. 가상환경 생성, 활성화 및 패키지 목록 설치
3. Article 모델 주석 해제 및 Migration 진행

```
# articles/models.py

class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

$ python manage.py migrate
```

- 준비된 fixtures 데이터 load

```
$ python manage.py loaddata articles.json
```

- DRF 설치, 등록 및 패키지 목록 업데이트

```
$ pip install djangorestframework
```

```
# settings.py

INSTALLED_APPS = [
    'articles',
    'django_extensions',
    'rest_framework',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
$ pip freeze > requirements.txt
```

ModelSerializer

ModelSerializer 작성

- `articles/serializers.py` 생성
 - `serializers.py`의 위치나 파일명은 자유롭게 작성 가능
- ModelSerializer 작성

```
# articles/serializers.py

from rest_framework import serializers
from .models import Article

class ArticleListSerializer(serializers.ModelSerializer):

    class Meta:
        model = Article
        fields = ('id', 'title', 'content',)
```

- ModelSerializer 클래스는 모델 필드에 해당하는 필드가 있는 Serializer 클래스를 자동으로 만들 수 있는 shortcut을 제공
 1. Model 정보에 맞춰 자동으로 필드를 생성

2. serializer에 대한 유효성 검사기를 자동으로 생성
3. `.create()` 및 `.update()`의 간단한 기본 구현이 포함됨

Serializer 연습하기

- `shell_plus` 실행 및 `ArticleListSerializer import`

```
$ python manage.py shell_plus
```

```
>>> from articles.serializers import ArticleListSerializer
```

- 인스턴스 구조 확인

```
>>> serializer = ArticleListSerializer()

>>> serializer
ArticleListSerializer():
    id = IntegerField(label='ID', read_only=True)
    title = CharField(max_length=10)
    content = CharField(style={'base_template': 'textarea.html'})
```

- Model instance 객체 serialize

```
>>> article = Article.objects.get(pk=1)

>>> serializer = ArticleListSerializer(article)

>>> serializer
ArticleListSerializer(<Article: Article object (1)>):
    id = IntegerField(label='ID', read_only=True)
    title = CharField(max_length=10)
    content = CharField(style={'base_template': 'textarea.html'})

# serialized data 조회
>>> serializer.data
{'id': 1, 'title': 'Site economic if two country science.' ...}
```

- QuerySet 객체 serialize

```

articles = Article.objects.all()

# many=True 옵션 X
>>> serializer = ArticleListSerializer(articles)
>>> serializer.data
AttributeError: Got AttributeError when attempting to get a value for field `title` on
serializer `ArticleListSerializer`.
The serializer field might be named incorrectly and not match any attribute or key on the
`QuerySet` instance.
Original exception text was: 'QuerySet' object has no attribute 'title'.

# many=True 옵션 O
>>> serializer = ArticleListSerializer(articles, many=True)
>>> serializer.data
[OrderedDict([('id', 1), ('title', 'Live left research.'), ('content', 'Small drive until back
board drive...')]]
```

ModelSerializer의 `many` option

- 단일 객체 인스턴스 대신 QuerySet 또는 객체 목록을 serialize 하려면 many=True 를 작성해야함

```

# many 예시

queryset = Book.objects.all()
serializer = BookSerializer(queryset, many=True)
serializer.data
# [
#     {'id': 0, 'title': 'The electric kool-aid acid test', 'author': 'Tom Wolfe'},
#     {'id': 1, 'title': 'If this is a man', 'author': 'Primo Levi'},
#     {'id': 2, 'title': 'The wind-up bird chronicle', 'author': 'Haruki Murakami'}
# ]
```

Build RESTful API - Article

URL과 HTTP requests methods 설계

	GET	POST	PUT	DELETE
articles/	전체 글 조회	글 작성	전체 글 수정	전체 글 삭제
articles/1/	1번 글 조회	.	1번 글 수정	1번 글 삭제

GET - List

- 게시글 데이터 목록 조회하기
- DRF에서 `api_view` 데코레이터 작성은 필수

```

# articles/urls.py

urlpatterns = [
    path('articles/', views.article_list),
]

# articles/views.py

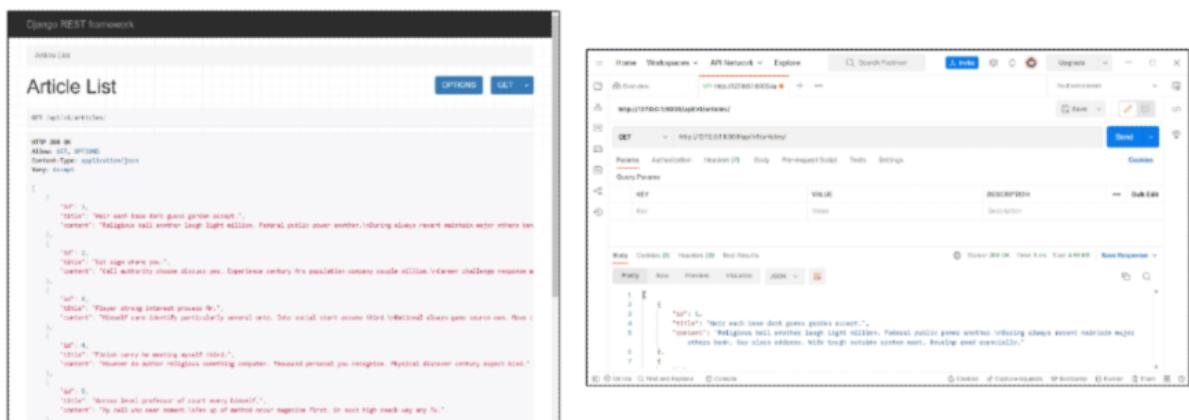
from rest_framework.response import Response
from rest_framework.decorators import api_view

from .models import Article
from .serializers import ArticleListSerializer

@api_view(['GET'])
def article_list(request):
    articles = Article.objects.all()
    serializer = ArticleListSerializer(articles, many=True)
    return Response(serializer.data)

```

- 응답 확인



api_view decorator

- DRF view 함수가 응답해야 하는 HTTP
- 기본적으로 GET 메서드만 허용되며 다른 메서드 요청에 대해서는 405 Method Not Allowed로 응답

GET - Detail

- 단일 게시글 데이터 조회하기
- 각 데이터의 상세 정보를 제공하는 ArticleSerializer 정의

```

# articles/serializers.py

class ArticleSerializer(serializers.ModelSerializer):

    class Meta:
        model = Article
        fields = '__all__'

```

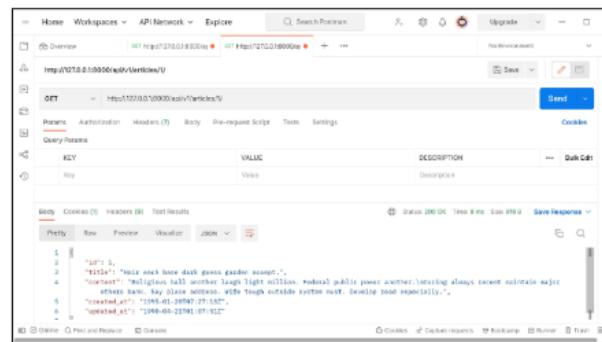
- url 및 view 함수 작성

```
# articles/urls.py
urlpatterns = [
    ...
    path('articles/<int:article_pk>', views.article_detail),
]
```

```
# articles/views.py
from .serializers import ArticleListSerializer, ArticleSerializer

@api_view(['GET'])
def article_detail(request, article_pk):
    article = Article.objects.get(pk=article_pk)
    serializer = ArticleSerializer(article)
    return Response(serializer.data)
```

- 응답 확인



POST

- 게시글 데이터 생성하기
- 요청에 대한 데이터 생성이 성공했을 경우는 201 Created 상태 코드를 응답하고 실패 했을 경우는 400 Bad request를 응답
- 요청에 대한 데이터 생성이 성공했을 경우는 201 Created 상태 코드를 응답하고 실패 했을 경우는 400 Bad request를 응답

```
# articles/views.py
from rest_framework import status

@api_view(['GET', 'POST'])
def article_list(request):
    if request.method == 'GET':
        articles = Article.objects.all()
        serializer = ArticleListSerializer(articles, many=True)
        return Response(serializer.data)

    elif request.method == 'POST':
        serializer = ArticleSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

- POST 응답 확인

Postman screenshot showing a successful POST request to `http://127.0.0.1:8000/api/v1/articles/`. The request method is highlighted in red. The 'Body' tab is selected, showing form-data fields `title` (value: 제목) and `content` (value: 내용). The response status is 201 Created with a JSON payload:

```

1
2   "id": 21,
3   "title": "제목",
4   "content": "내용",
5   "created_at": "2022-10-15T14:09:04.936667Z",
6   "updated_at": "2022-10-15T14:09:04.936667Z"
7

```

- 새로 생성된 데이터 확인 해보기

Django REST framework - Article Detail

Article Detail

GET /api/v1/articles/21/

```

HTTP 200 OK
Allow: OPTIONS, GET
Content-Type: application/json
Vary: Accept

{
    "id": 21,
    "title": "제목",
    "content": "내용",
    "created_at": "2022-10-15T14:09:04.936667Z",
    "updated_at": "2022-10-15T14:09:04.936667Z"
}

```

Raising an exception on invalid data

- 유효하지 않은 데이터에 대해 예외 발생시키기
- `is_valid()` 는 유효성 검사 오류가 있는 경우 ValidationError 예외를 발생시키는 선택적 `raise_exception` 인자를 사용할 수 있음

- DRF에서 제공하는 기본 예외 처리기에 의해 자동으로 처리되며 기본적으로 HTTP 400 응답을 반환
- view 함수 코드 변경

```
# articles/views.py

@api_view(['GET', 'POST'])
def article_list(request):
    ...
    elif request.method == 'POST':
        serializer = ArticleSerializer(data=request.data)
        if serializer.is_valid(raise_exception=True):
            serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
```

DELETE

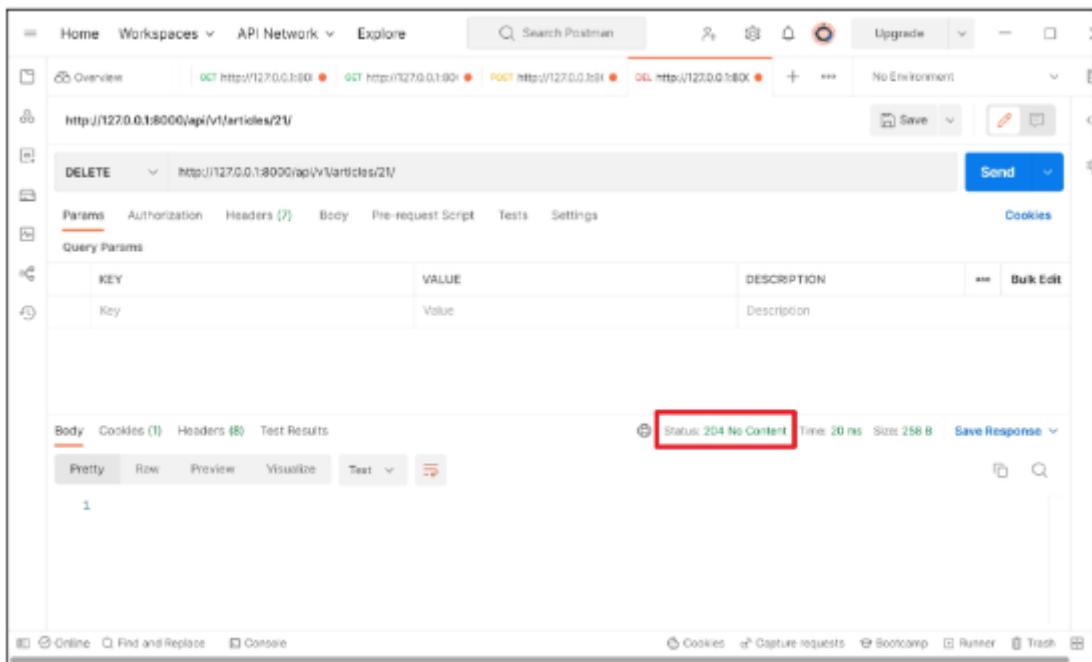
- 게시글 데이터 삭제하기
- 요청에 대한 데이터 삭제가 성공했을 경우는 204 No Content 상태 코드 응답(명령을 수행했고 더이상 제공할 정보가 없는 경우)

```
# articles/views.py

@api_view(['GET', 'DELETE'])
def article_detail(request, article_pk):
    article = Article.objects.get(pk=article_pk)
    if request.method == 'GET':
        serializer = ArticleSerializer(article)
        return Response(serializer.data)

    elif request.method == 'DELETE':
        article.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

- 응답 확인



PUT

- 게시글 데이터 수정하기
- 요청에 대한 데이터 수정이 성공했을 경우는 200 OK 상태 코드 응답

```
# articles/views.py

@api_view(['GET', 'DELETE', 'PUT'])
def article_detail(request, article_pk):
    ...
    elif request.method == 'PUT':
        serializer = ArticleSerializer(article, data=request.data)
        # serializer = ArticleSerializer(instance=article, data=request.data)
        if serializer.is_valid(raise_exception=True):
            serializer.save()
            return Response(serializer.data)
```

- PUT 응답 확인

The screenshot shows the Postman application interface. At the top, there are tabs for Home, Workspaces, API Network, and Explore. A search bar says "Search Postman". On the right, there are buttons for Upgrade, Minimize, and Close. Below the tabs, there's a navigation bar with icons for Overview, GET, POST, PUT, DELETE, and a plus sign. The "Overview" tab is selected, showing a status summary: 1 GET, 1 POST, 1 PUT, 1 DELETE, and 1223 total requests. To the right of the summary is a "No Environment" button. Under the "Overview" tab, the URL `http://127.0.0.1:8000/api/v1/articles/1/` is displayed, along with Save and Edit buttons.

In the main workspace, a "PUT" request is being configured. The URL is set to `http://127.0.0.1:8000/api/v1/articles/1/`. The "Body" tab is selected, showing the following JSON payload:

```
{  
    "id": 1,  
    "title": "제목 수정",  
    "content": "내용 수정",  
    "created_at": "1995-01-29T09:27:13Z",  
    "updated_at": "2022-10-15T14:40:38.833549Z"  
}
```

The "Body" tab also includes sections for Params, Authorization, Headers, Pre-request Script, Tests, and Settings. Under "Params", there are radio buttons for none, form-data, x-www-form-urlencoded, raw, binary, and GraphQL. The "form-data" option is selected. The "Body" table has columns for KEY, VALUE, and DESCRIPTION. It contains two rows: one for "title" with value "제목 수정" and one for "content" with value "내용 수정". There is also a "Bulk Edit" button at the bottom of the table.

Below the request configuration, there are tabs for Body, Cookies, Headers, and Test Results. The "Body" tab is selected. On the right, there are buttons for Status: 200 OK, Time: 20 ms, Size: 426 B, and Save Response. At the bottom, there are buttons for Pretty, Raw, Preview, Visualize, and JSON. The "Pretty" tab is selected. The JSON response is displayed as:

```
{  
    "id": 1,  
    "title": "제목 수정",  
    "content": "내용 수정",  
    "created_at": "1995-01-29T09:27:13Z",  
    "updated_at": "2022-10-15T14:40:38.833549Z"  
}
```

At the very bottom, there are links for Online, Find and Replace, Console, Cookies, Capture requests, Bootcamp, Runner, and Trash.

- 수정된 데이터 확인해보기

```
Django REST framework

Article List / Article Detail

Article Detail
DELETE OPTIONS GET ▾

GET /api/v1/articles/1/

HTTP 200 OK
Allow: PUT, GET, OPTIONS, DELETE
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "title": "제목 수정",
    "content": "내용 수정",
    "created_at": "1995-01-20T07:27:13Z",
    "updated_at": "2022-10-15T14:40:38.833549Z"
}
```

Django REST framework - N:1 Relation

개요

- N:1 관계에서의 모델 data를 Serialization하여 JSON 으로 변환하는 방법 학습

사전 준비

- Comment 모델 주석 해제 및 데이터 베이스 초기화

```
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

- Migration 진행

```
$ python manage.py makemigrations
$ python manage.py migrate
```

- 준비된 fixtures 데이터 load

```
$ python manage.py loaddata articles.json comments.json
```

GET - List

- 댓글 데이터 목록 조회하기
- Article List와 비교하며 작성해보기

```
# articles/serializers.py

from .models import Article, Comment

class CommentSerializer(serializers.ModelSerializer):

    class Meta:
        model = Comment
        fields = '__all__'
```

```

urlpatterns = [
    ...,
    path('comments/', views.comment_list),
]

# articles/views.py

from .models import Article, Comment
from .serializers import ArticleListSerializer, ArticleSerializer, CommentSerializer

@api_view(['GET'])
def comment_list(request):
    comments = Comment.objects.all()
    serializer = CommentSerializer(comments, many=True)
    return Response(serializer.data)

```

- GET 응답 확인

The image displays two side-by-side screenshots. On the left is a screenshot of the Django REST framework's built-in browsable API interface for the 'Comment List' endpoint. It shows a single comment object with fields: id, content, created_at, updated_at, and article. The content field contains a long, complex sentence. On the right is a screenshot of the Postman API testing tool. It shows a GET request to the same endpoint. The 'Body' tab of the Postman interface shows the JSON response received from the server, which is identical to the one shown in the Django interface. The response body is as follows:

```

{
    "id": 1,
    "content": "Twinkie free why name break. File routine become four. Really know good executive something improve. Later month star no",
    "created_at": "1990-01-01T00:00:00Z",
    "updated_at": "1990-01-01T00:00:00Z",
    "article": 20
},
{
    "id": 2,
    "content": "Material apply money believe. The smaller alone huge room hair covers. Billies family kitchen miss drop manage each mid",
    "created_at": "1990-01-01T00:00:00Z",
    "updated_at": "1990-01-01T00:00:00Z",
    "article": 5
}

```

GET - DETAIL

- 단일 댓글 데이터 조회하기
- Article과 달리 같은 serializer 사용하기

```
# articles/urls.py

urlpatterns = [
    ...,
    path('comments/<int:comment_pk>', views.comment_detail),
]

# articles/views.py

@api_view(['GET'])
def comment_detail(request, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    serializer = CommentSerializer(comment)
    return Response(serializer.data)
```

- GET 응답 확인

The screenshot shows the Django REST framework's "Comment Detail" view. The response status is "HTTP 200 OK". The response body contains JSON data:

```
{
    "id": 1,
    "content": "Finally free why name break. File receive become fear. Really break good executive something improve. Later month start how job",
    "created_at": "2023-11-07T11:00:00Z",
    "updated_at": "2023-11-07T11:00:00Z",
    "article": 20
}
```

Below the response, there is a detailed view of the API request in a browser-based tool like Postman. The URL is `http://127.0.0.1:8000/api/v1/comments/2/`. The method is GET. The Headers tab shows the Content-Type header set to application/json. The Body tab shows the same JSON data as the response.

POST

- 단일 댓글 데이터 생성하기

```
# articles/urls.py

urlpatterns = [
    ...,
    path('articles/<int:article_pk>/comments/', views.comment_create),
]

# articles/views.py

@api_view(['POST'])
def comment_create(request, article_pk):
    article = Article.objects.get(pk=article_pk)
    serializer = CommentSerializer(data=request.data)
    if serializer.is_valid(raise_exception=True):
        serializer.save()
    return Response(serializer.data, status=status.HTTP_201_CREATED)
```

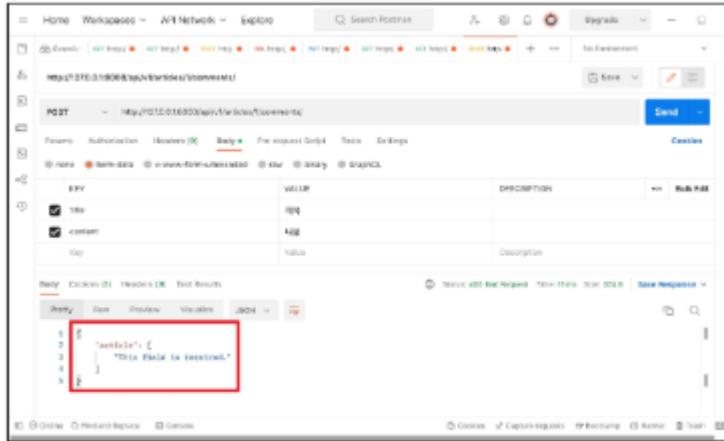
Passing Additional attributes to `.save()`

- `save()` 메서드는 특정 Serializer 인스턴스를 저장하는 과정에서 추가적인 데이터를 받을 수 있음
- `CommentSerializer` 를 통해 Serialize되는 과정에서 Parameter로 넘어온 `article_pk`에 해당하는 article 객체를 추가적인 데이터를 넘겨 저장

```
# articles/views.py

@api_view(['POST'])
def comment_create(request, article_pk):
    article = Article.objects.get(pk=article_pk)
    serializer = CommentSerializer(data=request.data)
    if serializer.is_valid(raise_exception=True):
        serializer.save(article=article)
    return Response(serializer.data, status=status.HTTP_201_CREATED)
```

- 응답 확인



- 에러 이유
 - CommentSerializer에서 article field 데이터 또한 사용자로부터 입력받도록 설정되어 있기 때문

읽기 전용 필드 설정

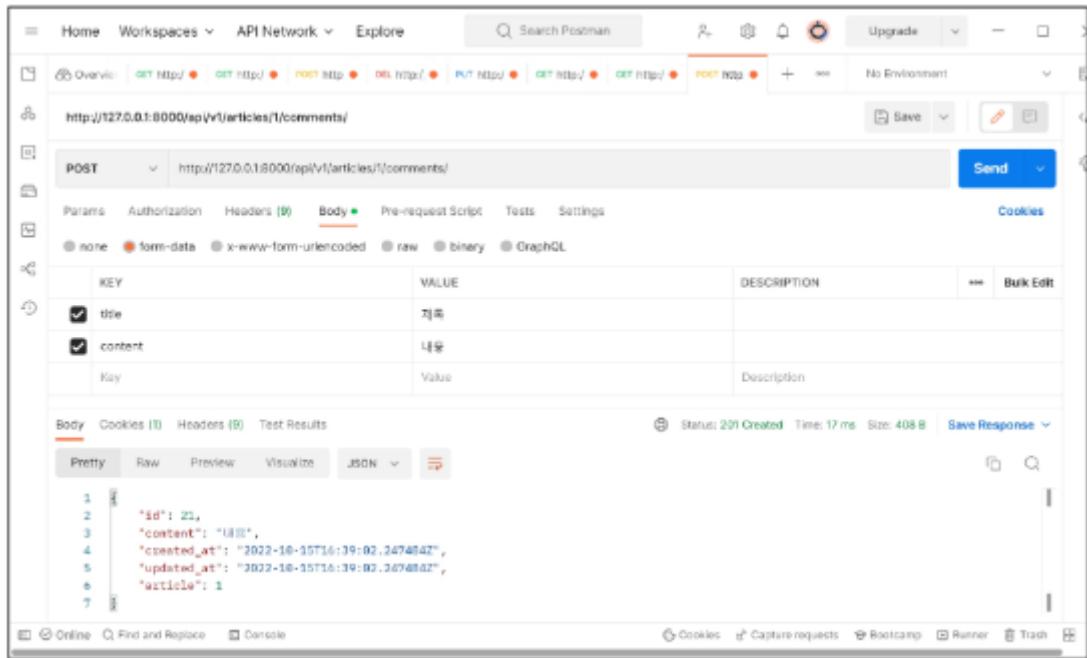
- `read_only_fields` 를 사용해 외래 키 필드를 읽기 전용 필드로 설정
- 읽기 전용 필드는 데이터를 전송하는 시점에 해당 필드를 유효성 검사에서 제외시키고 데이터 조회 시에는 출력 하도록 함

```
# articles/serializers.py

class CommentSerializer(serializers.ModelSerializer):

    class Meta:
        model = Comment
        fields = '__all__'
        read_only_fields = ('article',)
```

- POST 응답 재확인



DELETE & PUT

- 댓글 데이터 삭제 및 수정 구현하기

```
# articles/views.py

@api_view(['GET', 'DELETE', 'PUT'])
def comment_detail(request, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    if request.method == 'GET':
        serializer = CommentSerializer(comment)
        return Response(serializer.data)

    elif request.method == 'DELETE':
        comment.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

    elif request.method == 'PUT':
        serializer = CommentSerializer(comment, data=request.data)
        if serializer.is_valid(raise_exception=True):
            serializer.save()
            return Response(serializer.data)
```

- DELETE 응답 확인

The screenshot shows the Postman interface with a DELETE request to `http://127.0.0.1:8000/api/v1/comments/2/`. The response status is **204 No Content**, indicating the comment was successfully deleted.

- PUT 응답확인

The screenshot shows the Postman interface with a PUT request to `http://127.0.0.1:8000/api/v1/comments/1/`. The response status is **200 OK**, and the JSON response body is:

```

1 {
2     "id": 1,
3     "content": "내용 수정",
4     "created_at": "1975-12-07T13:38:25Z",
5     "updated_at": "2022-10-15T16:40:49.261377Z",
6     "article": 29
7 }
  
```

N:1 - 역참조 데이터 조회

개요

1. 특정 게시글에 작성된 댓글 목록 출력하기

- a. 기존 필드 override

2. 특정 게시글에 작성된 댓글의 개수 출력하기

a. 새로운 필드 추가

1. 특정 게시글에 작성된 댓글 목록 출력하기

- 기존 필드 override - Article Detail
 - 게시글 조회 시 해당 게시글의 댓글 목록까지 함께 출력하기
 - Serializer는 기존 필드를 override하거나 추가적인 필드를 구성할 수 있음
- `PrimaryKeyRelatedField()`

```
# articles/serializers.py

class ArticleSerializer(serializers.ModelSerializer):
    comment_set = serializers.PrimaryKeyRelatedField(many=True, read_only=True)

    class Meta:
        model = Article
        fields = '__all__'
```

- 댓글이 있는 게시글 응답 예시

The screenshot displays two parts. On the left is the Django REST framework's Article Detail view, showing an article with its ID, title, content, and a 'comment_set' field containing a list of comment IDs (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). On the right is a Postman API request for the same endpoint, showing the JSON response with the 'comment_set' field expanded to show individual comment objects.

- `models.py`에서 `related_name`을 통해 이름 변경 가능
- 역참조 시 생성되는 `comment_set`을 override 할 수 있음

```
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name='comments')
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

- 작성 후 삭제

2. Nested relationships

```
# articles/serializers.py

class CommentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Comment
        fields = '__all__'
        read_only_fields = ('article',)

class ArticleSerializer(serializers.ModelSerializer):
    comment_set = CommentSerializer(many=True, read_only=True)

    class Meta:
        model = Article
        fields = '__all__'
```

- 모델 관계 상으로 참조된 대상은 참조하는 대상의 표현에 포함되거나 중첩(nested)될 수 있음
- 이러한 중첩된 관계는 serializers
- 댓글이 있는 게시글 응답 예시

The left screenshot shows the Django Admin 'Article Detail' page for article ID 1. It displays the article's title, content, and a list of comments. One comment is expanded, showing its full text and a reply from another user.

The right screenshot shows the Postman API client. A GET request is made to `/api/articles/1/`. The response body shows the JSON representation of the article, which includes a `comment_set` field containing the details of the comments associated with the article.

2. 특정 게시글에 작성된 댓글의 개수 출력하기

- 새로운 필드 추가 - Article Detail
 - 게시글 조회 시 해당 게시글의 댓글 개수까지 함께 출력하기

```
# articles/serializers.py

class ArticleSerializer(serializers.ModelSerializer):
    comment_set = CommentSerializer(many=True, read_only=True)
    comment_count = serializers.IntegerField(source='comment_set.count', read_only=True)

    class Meta:
        model = Article
        fields = '__all__'
```

- **source**

- serializers field's argument
- 필드를 채우는데 사용할 속성의 이름
- 점 표기법(dotted notation)을 사용하여 속성을 탐색할 수 있음

- 댓글이 있는 게시글 응답 예시

The screenshot shows two side-by-side interfaces. On the left is a browser window titled 'Article Detail' showing JSON data for an article. On the right is the Postman application interface showing a successful GET request to 'http://127.0.0.1:8000/api/v1/articles/1/'. The response body contains the same JSON data as the browser, including the 'comment_set' field.

137

🚧 읽기 전용 필드 지정 이유

- 특정 필드를 override 혹은 추가한 경우 `read_only_fields` 가 동작하지 않으니 주의

```
# 사용 불가능

class ArticleSerializer(serializers.ModelSerializer):
    comment_set = CommentSerializer(many=True)
    comment_count = serializers.IntegerField(source='comment_set.count')

    class Meta:
        model = Article
        fields = '__all__'
        read_only_fields = ('comment_set', 'comment_count',)
```

Django shortcuts functions

개요

- `django.shortcuts` 패키지는 개발에 도움될 수 있는 여러 함수와 클래스를 제공
- 제송되는 shortcuts 목록
 - `render()`, `redirect()`, `get_object_or_404()`, `get_list_or_404()`

`get_object_or_404()`

- 모델 manager objects에서 `get()` 을 호출하지만, 해당 객체가 없을 땐 기존 DoesNotExist 예외 대신 Http404를 raise함

```
# articles/views.py

from django.shortcuts import get_object_or_404

article = Article.objects.get(pk=article_pk)
comment = Comment.objects.get(pk=comment_pk)

# 위 코드를 모두 다음과 같이 변경
article = get_object_or_404(Article, pk=article_pk)
comment = get_object_or_404(Comment, pk=comment_pk)
```

`get_list_or_404()`

- 모델 manager objects에서 `filter()` 의 결과를 반환하고 해당 객체 목록이 없을 땐 Http404를 raise함

```
# articles/views.py

from django.shortcuts import get_object_or_404, get_list_or_404

articles = Article.objects.all()
comments = Comment.objects.all()

# 위 코드를 모두 다음과 같이 변경
articles = get_list_or_404(Article)
comments = get_list_or_404(Comment)
```

적용 전/후 비교

- 존재하지 않는 게시글 조회 시 이전에는 500 상태코드를 응답했지만 현재는 404 상태코드를 응답

작성 전

작성 후

왜 사용해야 할까?

- 클라이언트 입장에서 서버에 오류가 발생하여 요청을 수행할 수 없다(500)라는 원인이 정확하지 않은 에러를 마주하기보다는, 서버가 적절한 예외 처리를 하고 클라이언트에게 올바른 에러를 전달하는 것 또한 중요한 요소