

REST API 및 Django REST Framework(DRF) 학습 정리

1. REST API

API

- **API 정의 (Application Programming Interface):** 두 소프트웨어가 서로 통신할 수 있게 만드는 매커니즘입니다.
 - 클라이언트-서버처럼 서로 다른 프로그램에서 요청과 응답을 받을 수 있도록 만든 체계이며, 서로 이해할 수 있는 공통된 **규칙과 형식**을 제공합니다.
- **API 역할:**
 - 중요한 내부의 것을 감추고(캡슐화), 외부에 필요한 플러그인(기능)만 제공합니다.
 - 재사용성이 높은 코드를 대신 수행하여 복잡한 코드를 단순화합니다.
 - (예시) 날씨 앱이 기상청 서버에 정해진 형식으로 요청하면, 기상청이 그 형식에 맞춰 데이터를 전달해 주는 것.

REST API

- **REST 정의 (Representational State Transfer):** API 서버를 개발하기 위한 일종의 **소프트웨어 설계 방법**론입니다. 엄격한 규칙을 의미하는 것은 아니며, REST의 원리를 따르는 시스템을 **RESTful**하다고 부릅니다.
- **핵심 원리:** 자원(Resource), 자원의 행위(Verb), 자원의 표현(Representation)을 사용합니다.
 - **자원 (Resource):** 웹 서버가 관리하는 데이터나 대상 (예: 게시물, 사용자).
 - **자원의 주소:** 자원을 식별하는 방법으로 ****URI (Uniform Resource Identifier)****를 사용합니다.
 - **자원의 행위:** 자원에 대한 행위를 **HTTP Method**로 나타냅니다.

자원의 식별 (URI/URL)

- **URI (Uniform Resource Identifier):** 인터넷에서 자원(리소스)을 식별하는 문자열입니다. 가장 일반적인 형태는 **URL**입니다.
- **URL (Uniform Resource Locator):** 웹에 주어진 리소스의 주소, 즉 네트워크 상에 리소스가 어디에 있는지 알려주기 위한 약속입니다.
- **URL 구성 요소:**
 - **Scheme (프로토콜):** 브라우저가 리소스를 요청하는 데 사용해야 하는 규약 (예: `http` , `https`).
 - **Domain Name:** 요청 서버를 나타냅니다 (예: `www.example.com`).
 - **Port:** HTTP-80, HTTPS-443 등이며, 생략 가능합니다.
 - **Path:** 웹 서버의 리소스 경로를 나타냅니다. 실제 물리적 파일 위치가 아닌 추상적인 형태로 표현됩니다. (예: `/path/to/myfile.html` , `/articles/`)
 - **Anchor (#):** 문서 내 특정 위치(북마크)를 나타냅니다. 서버에 전달되지 않고 브라우저가 직접 이동할 수 있도록 합니다.

자원의 행위 (HTTP Methods)

자원에 대해 수행할 동작을 HTTP Method로 나타냅니다.

| **HTTP Method** | **자원의 행위** | **설명** || **GET** | 조회 (Read) | 리소스의 표현을 요청합니다. 조회 시 데이터는 검색해야 합니다. || **POST** | 생성 (Create) | 요청된 리소스에 새로운 리소스를 생성하고 이를 요청합니다. || **PUT** | 수정 (Update) | 리소스의 전체를 수정합니다. 데이터 전체를 전달해야 합니다. || **PATCH** | 수정 (Update) | 리소스의 일부를 수정합니다. 일부 필드만 전달해도 됩니다. || **DELETE** | 삭제 (Delete) | 리소스를 삭제합니다. |

- **HTTP Response Status Codes:** 요청이 성공적으로 완료되었는지 여부를 나타냅니다.
 - 1XX (Informational responses): 요청 처리 중
 - 2XX (Successful responses): 요청 성공 (200 OK, 201 Created 등)
 - 3XX (Redirection messages): 다른 위치로 옮겨짐
 - 4XX (Client error responses): 클라이언트 오류 (400 Bad Request, 404 Not Found 등)
 - 5XX (Server error responses): 서버 오류 (500 Internal Server Error 등)

자원의 표현 (Representation) - JSON 데이터 응답

- **JSON (JavaScript Object Notation):** 데이터를 구조화된 텍스트 형태로 표현하는 최소한의 형식입니다.
 - 클라이언트와 서버가 JSON 타입으로 데이터를 주고받을 수 있으며, 이는 언어와 플랫폼에 독립적으로 통신할 수 있는 기반이 됩니다.
 - REST API 서버는 HTML 페이지를 만들지 않고, **JSON 데이터**만 응답하여 응답 속도가 빨라집니다.

2. DRF with Single Model

Django REST framework (DRF)

- **정의:** Django에서 RESTful API 서버를 쉽게 구축할 수 있도록 도와주는 **오픈소스 소프트웨어 라이브러리**입니다.

Serializer (직렬화/역직렬화)

- **Serialization (직렬화):** 여러 시스템에서 활용하기 위해 **데이터 구조나 객체 상태를 재구성할 수 있는 포맷 (JSON)으로 변환하는 과정**입니다.
 - DRF에서는 Python 객체(Model Instance)를 JSON 형태로 변환하는 역할을 합니다.
- **Deserialization (역직렬화):** JSON 등의 데이터 포맷을 다시 Python 객체(Model Instance)로 복원하는 과정입니다.
- **Serializer 클래스 역할:** 클라이언트의 입력 데이터를 받아 유효성을 검사하고, Model Instance로 변환하는 핵심 역할을 합니다.

ModelSerializer

- **사용:** Django 모델을 기반으로 Serializer를 정의할 때 사용하며, 모델 필드에 맞추어 Serializer 필드를 자동으로 생성해줍니다.
- **사용 예시:**

```
from rest_framework import serializers
from .models import Article

class ArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Article
        fields = '__all__' # 모델의 모든 필드를 사용
```

3. CRUD with ModelSerializer

HTTP Method를 사용하여 데이터 조회(GET), 생성(POST), 수정(PUT/PATCH), 삭제(DELETE) 작업을 수행하는 방법입니다.

GET method - 조회

- **데이터 목록 조회:**
 - ModelSerializer의 `many=True` 속성을 사용하여 복수 개의 객체를 직렬화합니다.
 - `Article.objects.all()` 로 전체 Article 객체들을 가져옵니다.
 - 뷰 함수에서 `@api_view(['GET'])` 데코레이터를 사용하여 GET 요청만 받도록 설정합니다.

```
# List 조회 (articles/views.py)
@api_view(['GET'])
def article_list(request):
    articles = Article.objects.all()
    serializer = ArticleSerializer(articles, many=True) # many=True로 목록 조회
    return Response(serializer.data) # .data에 직렬화된 JSON 데이터 포함
```

- **단일 데이터 조회:**
 - URL Path에서 Primary Key(`pk`)를 받아 특정 객체를 조회합니다.
 - ModelSerializer를 단일 객체에 적용합니다.

POST method - 생성

- `@api_view(['POST'])` 데코레이터를 사용합니다.
- 클라이언트로부터 받은 데이터를 Serializer에 전달하고 유효성 검사를 수행합니다.
 - `serializer = ArticleSerializer(data=request.data)` : 클라이언트의 데이터를 Serializer에 전달합니다.
 - `if serializer.is_valid()` : 데이터 유효성 검사를 수행합니다.
 - `serializer.save()` : 유효성 검사를 통과하면 DB에 객체를 생성하고 저장합니다.
- **성공 응답:** `Response(serializer.data, status=status.HTTP_201_CREATED)` (201 Created)를 반환합니다.
- **실패 응답:** `Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)` (400 Bad Request)를 반환합니다.

DELETE method - 삭제

- `@api_view(['DELETE'])` 데코레이터를 사용합니다.
- URL Path에서 `pk` 를 받아 특정 객체를 찾습니다.
- `article.delete()` 를 통해 DB에서 객체를 삭제합니다.

- **성공 응답:** `Response(status=status.HTTP_204_NO_CONTENT)` (204 No Content)를 반환하며, 삭제 성공 시 본문(body)에 데이터가 없음을 나타냅니다.

PUT method - 수정 (전체 수정)

- `@api_view(['PUT'])` 데코레이터를 사용합니다.
- URL Path에서 `pk` 를 받아 특정 객체를 찾습니다.
- **Serializer 인스턴스에 기존 객체와 데이터를 모두 전달해야 합니다.**
 - `serializer = ArticleSerializer(article, data=request.data):` `article` (기존 객체)과 `data` (수정 데이터)를 전달합니다.
- POST와 동일하게 유효성 검사 후 `serializer.save()` 를 호출하면, **기존 객체를 덮어쓰기**하여 수정합니다.

PATCH method - 수정 (부분 수정)

- **PUT vs PATCH:**
 - **PUT:** 리소스의 전체를 교체하며, 모든 필수 필드를 포함해야 합니다. (전체 수정)
 - **PATCH:** 리소스의 일부 필드만 수정하며, 수정할 필드만 전달해도 됩니다. (부분 수정)
- **DRF에서 PATCH 사용:**
 - Serializer 인스턴스 생성 시 `partial=True` 인자를 추가합니다.
 - `serializer = ArticleSerializer(article, data=request.data, partial=True)`
 - 이 경우, 필수 필드가 없더라도 오류를 발생시키지 않고 전달된 필드만 수정할 수 있습니다.

4. 참고

`raise_exception`

- `serializer.is_valid()` 의 선택 인자입니다.
- `raise_exception=True` 로 설정하면, 유효성 검사에 실패할 경우 DRF가 자동으로 `Validation Error` 를 발생시키고 이를 HTTP 400 응답으로 변환하여 반환합니다.
- 이로써 개발자가 별도로 `if/else` 구문으로 오류 처리를 명시하지 않아도 됩니다.

```
# raise_exception 예시
if request.method == 'POST':
    # raise_exception=True 설정으로 유효성 실패 시 자동 400 Bad Request 응답
    serializer = ArticleSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    serializer.save()
    return Response(serializer.data, status=status.HTTP_201_CREATED)
```

Generated by Gemini