# RESTful API

## REST? API?

RESTful API에서 REST는 **RE**presentational **S**tate **T**ransfer의 약자로, 웹과 같은 분산 하이퍼미디어 시스템을 위한 아키텍쳐 스타일을 의미합니다. API는 **A**pplication **P**rogramming **I**nterface의 약자로, 다른 소프트웨어 시스템과 통신하기 위해 따라야 하는 규칙을 정의합니다. 따라서 RESTful API는 *REST 아키텍쳐 스타일을 따르는 API*를 의미합니다.

## RESTful API의 동작

RESTful API가 동작하는 방식은 다음과 같습니다. 먼저 클라이언트가 API 문서에 따라 서버가 이해하는 형식으로 요청을 전송합니다. 서버에서는 클라이언트를 인증하고 권한을 확인한 후, 요청을 내부적으로 처리합니다. 마지막으로 서버에서 요청을 성공적으로 처리했는지, 클라이언트가 요청한 정보와 함께 응답을 반환합니다.

## RESTful API의 요청

RESTful API의 요청에는 고유 리소스 식별자(URI), 메서드, 헤더, 데이터, 파라미터가 포함됩니다. HTTP를 이용해 구현한다는 가정하에 GET, POST, PUT, PATCH, DELETE를 포함해 9개의 HTTP 메서드가 존재합니다.

또한 Microsoft에서 작성한 RESTful Web API Design 문서에서 권장하는 RESTful API 네이밍 가이드라인은 다음과 같습니다.

1. 리소스는 명사로 표현한다. HTTP 메서드가 동사의 역할을 하기 때문에 URI에 동사를 사용하지 않는다.
2. 소문자만을 이용한다.
3. 리소스간의 관계를 계층적으로 표현하여 명확한 구조를 만든다.
   *예) /users/{id}/orders/{orderId} : 특정 사용자의 특정 주문 상세정보*
4. 필터링과 검색은 URI 매개변수를 통해 처리한다.
5. 날짜 형식은 ISO 8601을 따른다.
6. 여러 단어로 구성된 리소스를 표현할 때 하이픈(-)을 사용한다.
7. API의 버전은 URI에 포함해 명시한다.
   *예) /v1/users*
8. 적절한 HTTP 상태코드(200, 201, 400, 401, 404, 500 등)를 반환한다.

# CRUD API

CREATE(생성), READ(읽기), UPDATE(수정), DELETE(삭제)의 기능을 수행할 수 있는 API를 RESTful API의 컨벤션에 맞게 구현한다. Spring Boot와 PostgreSQL(Supabase)를 이용해 구현하였다.

## 책 정보 생성하기 (CREATE)

POST /books HTTP/1.1
Host: localhost:8080
Content-Type: application/json

```
{
   "name": "도서1",
   "author": "저자1",
   "publisher": "출판사1",
   "summary": "도서 요약1",
}
```

HTTP/1.1 201
Content-Type: application/json

```
{
   "id": 4,
   "name": "도서1",
   "author": "저자1",
   "publisher": "출판사1",
   "summary": "도서 요약1",
   "createdAt": "2024-09-29T22:34:51.817511",
   "updatedAt": "2024-09-29T22:34:51.817511"
}
```

## 책 목록 가져오기 (READ)

GET /books HTTP/1.1
Host: localhost:8080

```
HTTP/1.1 200
Content-Type: application/json


[
  {
    "id": 4,
    "name": "도서1",
    "author": "저자1",
    "publisher": "출판사1",
    "summary": "도서 요약1",
    "createdAt": "2024-09-29T22:34:51.817511",
    "updatedAt": "2024-09-29T22:34:51.817511"
  }, ...
]
```

## 책 정보 읽기 (READ)

```
GET /books/{id} HTTP/1.1
Host: localhost:8080


HTTP/1.1 200
Content-Type: application/json


{
  "id": 4,
  "name": "도서1",
  "author": "저자1",
  "publisher": "출판사1",
  "summary": "도서 요약1",
  "createdAt": "2024-09-29T22:34:51.817511",
  "updatedAt": "2024-09-29T22:34:51.817511"
}
```

## 책 정보 갱신하기 (UPDATE)

PUT /books/{id} HTTP/1.1
Host: localhost:8080

```
{
   "name": "도서 수정1",
   "author": "저자 수정1",
   "publisher": "출판사 수정1",
   "summary": "도서 요약 수정1"
}
```

HTTP/1.1 200
Content-Type: application/json

```
{
   "id": 4,
   "name": "도서 수정1",
   "author": "저자 수정1",
   "publisher": "출판사 수정1",
   "summary": "도서 요약 수정1",
   "createdAt": "2024-09-29T22:34:51.817511",
   "updatedAt": "2024-09-29T22:36:51.003522"
}
```

## 책 정보 삭제하기 (DELETE)

DELETE /books/{id} HTTP/1.1
Host: localhost:8080

HTTP/1.1 200

# 개선할 점

RESTful API의 구성을 상세히 알아보기 전에 구현을 진행하고, RESTful API에 대해 상세히 학습해보았다. 따라서 각각의 HTTP 메소드에 대해 명확한 응답 코드를 응답하지 못하고 있는 것 같아 이에 대한 수정이 필요할 것으로 보인다. 특히 PUT, DELETE 요청에 대한 처리 방법에 대해 더 학습하고 코드를 개선해 나갈 필요가 있다. 또한 PUT 요청과 PATCH 요청의 차이(idempotent)를 학습하고 UPDATE 요청에 대한 처리를 조금 더 상세히 변경할 필요가 있어 보인다.

또한 API 문서 작성과 데이터베이스 설계에 대해서도 고민해볼 필요가 있다. API 문서를 표로 정리하는 것도 좋은 방법일 것 같다. 데이터베이스 테이블을 Book 테이블과 Author 테이블로 나누어 더욱 세부적인 정보들을 다루고 매핑 관계 설정을 통해 데이터를 가져오도록 추가 구현해볼 필요가 있다.

POST, PUT, DELETE 요청에 대해 인증과 인가를 도입하는 것도 중요할 것으로 보인다. 현재는 간단히 CRUD가 가능한 API를 구현하였지만, 추후에는 관리자만이 수정 및 변경이 가능하도록 Spring Security를 이용해 추가 구현할 필요가 있다.

# 정말 RESTful API인가?

Naver D2에서 발표된 "그런 REST API로 괜찮은가"라는 발표에서 REST의 정의와 현대 RESTful API에서 잘 지켜지지 못하는 스타일에 대해 설명한다.

REST를 구성하는 스타일은 다음 6개의 항목이 있다.

- client-server
- stateless
- cache
- **uniform interface**
- layered system
- code-on-demand (optional)

대부분은 HTTP를 사용하기 때문에 충족되나, **uniform interface**가 지켜지지 않는 경우가 많다. 이 스타일의 세부 내용을 나열하면 다음 4개와 같다.

- identification of resources
- manipulation of resources through representations
- **self-descriptive messages:**
  **메세지가 스스로를 설명해야 한다. 목적지(Host), Content-Type 등이 정의되어 헤더를 통해 메세지를 해석할 수 있어야함.**

- **hypermedia as the engine of application state (HATEOAS):**
  **애플리케이션의 상태는 Hyperlink를 이용해 전이되어야한다. HTML의 Anchor태그, Link 헤더를 통해 표현 가능.**

Self-descriptive message와 HATEOAS가 잘 지켜지지 않기 때문에 REST API가 Roy Fielding이 정의한대로 작성되지 않는다. 서버와 클라이언트가 독립적으로 진화하기 때문에 서버의 기능이 변경되어도 클라이언트를 업데이트할 필요가 없어야 한다고 이야기한다. Roy Fielding의 REST API에 대한 정의는 <하이퍼텍스트를 포함한 self-descriptive한 메시지의 uniform interface를 통해 리소스에 접근하는 API>라고 할 수 있다.

다만, 내부적으로 사용되는 API와 같은 상황에선 Roy Fielding이 정의한대로 REST API가 작성되지 못할 수 있으므로 너무 REST API의 스타일에 고정될 필요는 없다고 한다. 하지만 여전히 REST 아키텍쳐 스타일을 지키지 못했기 때문에 REST API라고 부를 수는 없다고 한다.

# 출처


- RESTful API란?, AWS (https://aws.amazon.com/ko/what-is/restful-api)
- HTTP 요청 메서드, MDN (https://developer.mozilla.org/ko/docs/Web/HTTP/Methods)
- 그런 REST API로 괜찮은가, Naver D2 (https://www.youtube.com/watch?v=RP_f5dMoHFc)