RESTful

REST, RESTful, RESTful API

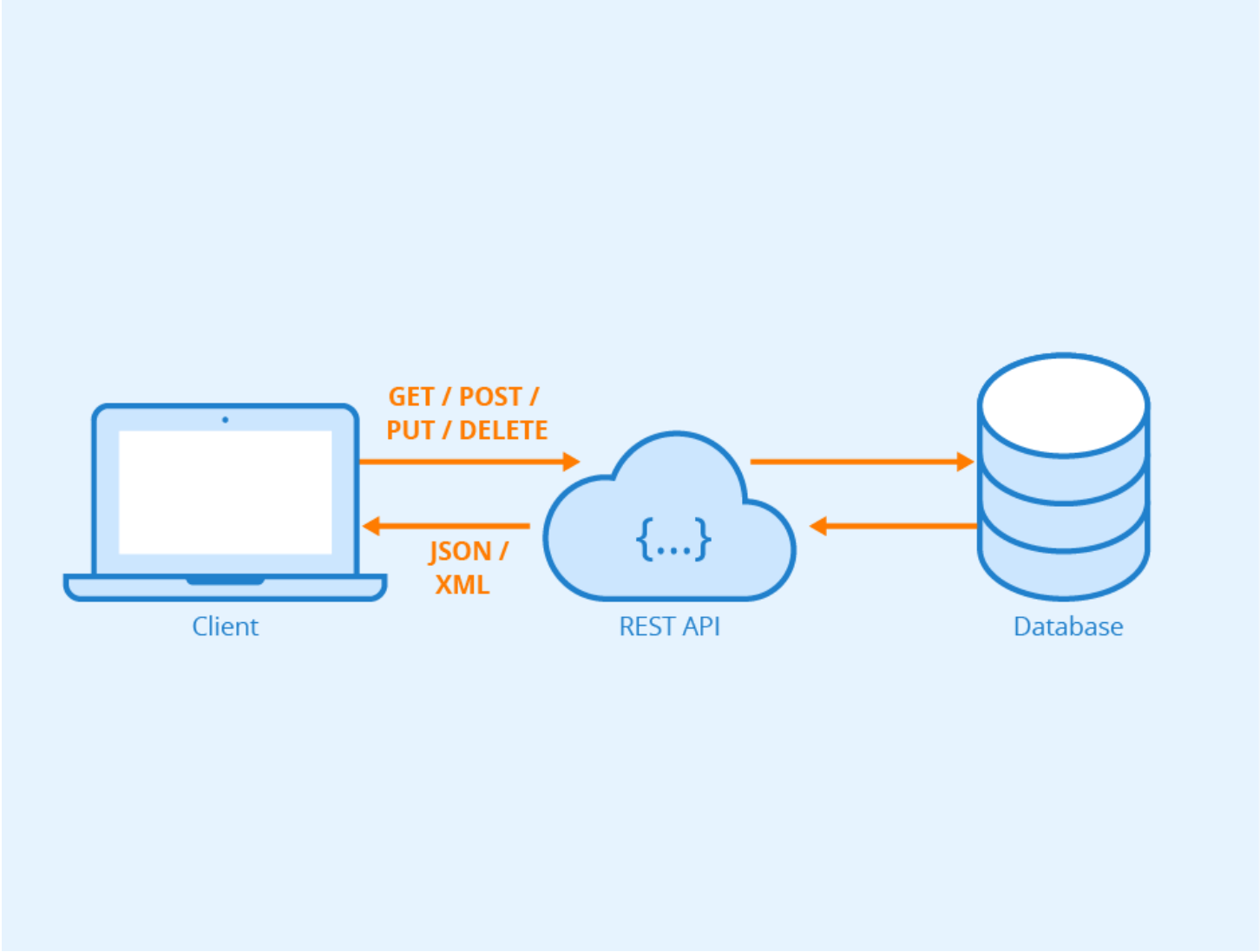
발표자 : 최윤제
2025-02-23

RESTful

- REST 아키텍쳐 스타일의 원칙을 준수하는 것.
- REST를 지키는 시스템 => **"RESTful API"**
- REST API를 제공하는 웹 서비스를 **RESTful**하다고 함
- REST를 REST처럼 쓰기 위한 방법, 누군가 공식적으로 정한 것은 아님

REST(Representational State Transfer) API

- REST : www 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식
- 자원을 이름으로 구분하여 해당 자원의 상태(정보)를 주고 받는 모든 것을 의미.
- HTTP 활용
  >웹의 장점 최대한 활용가능

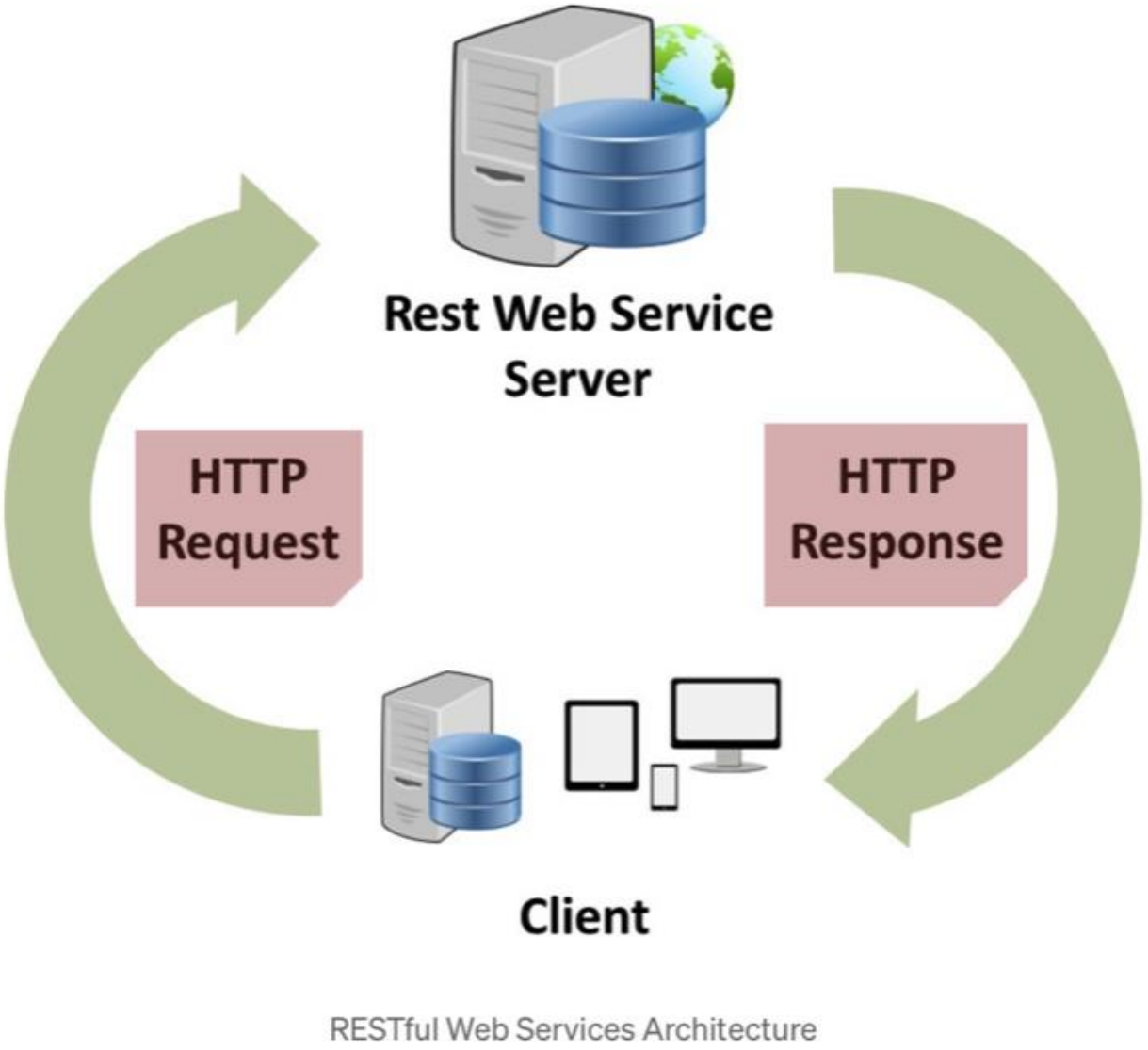


REST

- HTTP URI을 통해 자원을 명시
- HTTP Method를 통해 해당 자원에 대한 처리 가능

⇒REST란 결국 자원, 리소스 기반 구조 설계

**웹의 모든 자원에 고유한 ID(주소)인 HTTP URI를 부여!



RESTful Web Services Architecture

REST의 구성(1)

- **자원, 행위, 표현**으로 나뉨

1. 자원 - URL
=>모든 자원은 서버에 있고, 고유한 ID가 존재함

자원을 구별하는 ID는 **HTTP URI**이다.
(HTTP URI는 그림과 같은 형식)

`/orders/order_id/1`

- **리소스 컬렉션**: `/users`
- **단일 리소스**: `/users/{id}`

REST의 구성(2)

2. 행위 - HTTP Method

=>HTTP의 메소드를 사용. POST, GET, PUT, DELETE 등.

(C, R, U, D)

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

REST의 구성(3)

3. 표현

=>클라이언트가 자원의 상태에 대해 조작을 요청하면, 서버는 이에 적절한 응답을 보냄

적절한 응답에는 JSON, XML, TEXT, RSS등 여러 형태로 나타냄

*현재는 대부분 JSON으로 주고 받음

REST 특징

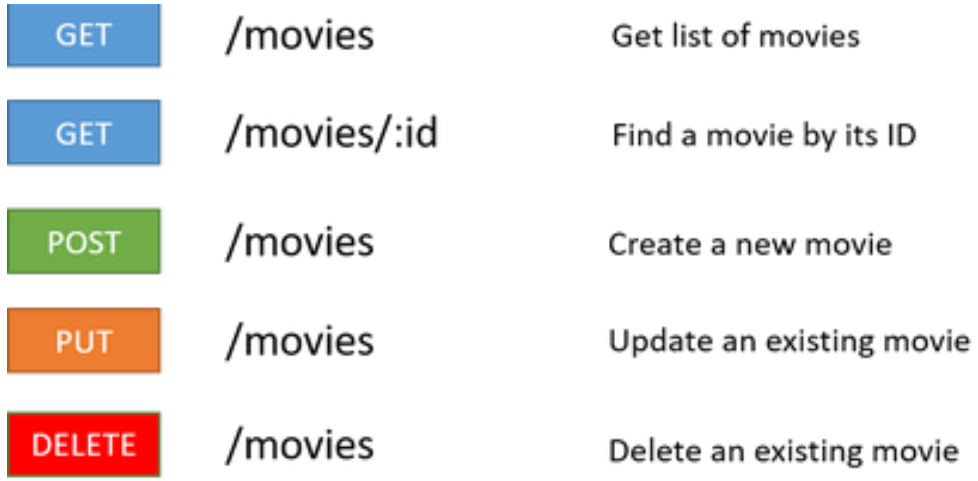
- 클라이언트 / 서버 구조
- 무상태성(Stateless) – 상태를 기억할 필요가 없음
- 캐시 처리 가능(Cacheable)
- 자체 표현 구조(Self – descriptiveness)
- 계층화
- 유니폼 인터페이스 – **특정 언어나 기술에 종속되지 않음**

REST의 규칙

중심 규칙

-**URI는 정보의 자원을 표현**해야 한다

-자원에 대한 **행위는 HTTP Method**로 표현한다



세부 규칙

-'/'는 계층 관계를 나타내는데 사용

-URI 마지막 문자로 '/'를 포함하지 않음

-소문자 사용, '-'으로 가독성 높임, '_' 사용 금지

-URI에 파일확장자 포함 X -> 대신 Accept: 형태의 헤더 사용

-등등 자세한건 출처 블로그 보기

REST의 핵심 설계 목표

- **상호연동성 확보**

  -다른 컴포넌트들을 쉽게 연결할 수 있는 성질

  -두 개이상의 컴포넌트를 결합함으로써 작업을 효율적으로 수행

- **범용 인터페이스**

  -상호연동성처럼 어디서든 사용가능한 범용 인터페이스 제공

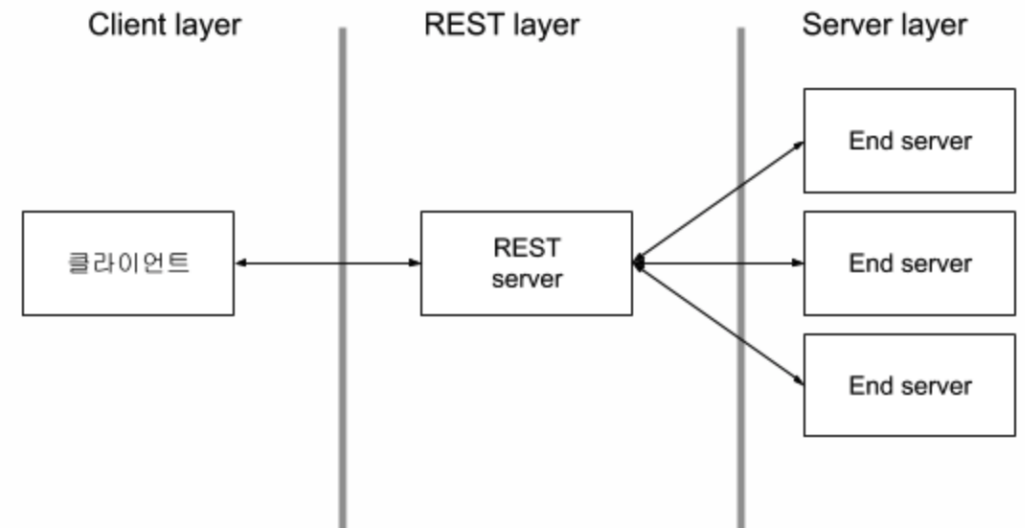
- **각 컴포넌트들의 독립적인 배포**

  -다른 컴포넌트들과 독립적으로 개발 가능

- **컴포넌트를 중계하는 역할**

  -REST 서버가 클라이언트와 서버 중간에서 중계역할

  => 확장성, 성능 향상 및 보안 정책 적용에도 용이



RESTful API 개발 원칙

- **자원 식별이 가능해야한다.**

- **행위가 명시적일 것.**

- **자기 서술적일 것.**

  -데이터 처리를 위한 정보를 얻기 위해 데이터 원본을 읽어야 한다면 자기 서술적이지 못하다고 봄

- **HATEOAS** (Hypermedia as the Engine of Application State)

  -클라이언트 요청에 대해 응답을 할때 추가적인 정보를 제공하는 링크(하이퍼 링크)를 포함할 수 있어야 한다.

REST 단점

- REST는 point-to-point 방식으로 **서버와 클라이언트가 연결을 맺고 상호작용해야하는 어플리케이션의 개발에는 부적합**함.
- REST는 URI와 HTTP를 이용한 아키텍처링 방법에 대한 내용만 가지고 있어, **개발자가 통신과 정책에 대한 설계와 구현을 도맡아야 한다.**
- HTTP에 상당히 의존적이다.
- CRUD 4가지 메소드만 제공한다.

출처

- https://velog.io/@somday/RESTful-API-%EC%9D%B4%EB%9E%80

- https://velog.io/@s0nnyday/RESTful-API%EC%9D%98-%EC%9D%B4%ED%95%B4%EC%99%80-%EC%84%A4%EA%B3%84-%EC%9B%90%EC%B9%99

왜 RESTful API??

1. 클라이언트를 정형화된 플랫폼으로 고정한 것이 아니라 PC, 모바일, 어플리케이션 등 플랫폼 제약을 두지 않기 때문에!

2. 메시지 기반, XML, JSON과 같은 클라이언트에서 바로 객체로 치환 가능한 형태의 데이터 통신을 지향 -> 서버와 클라이언트의 역할이 분리!

=>HTTP 표준 규약을 지키면서 API를 만드는 것이 중요해짐!

*클라이언트가 정형화 되지 않은 환경에서 표준을 지키지 않으면 개발 속도가 저하됨