REST API

REST란

로이 필딩이 발표한 아키텍처로 Representational State Transfer의 약자이며, HTTP URI를 통해 **자원(Resource)**을 명시하고, **HTTP Method** (GET, POST, PUT, DELETE)를 통해 해당 자원에 대한 **CRUD Operation**을 적용하는 것을 의미하며, REST의 기본 원칙을 성실히 지킨 서비스 디자인을 "RESTful"하다고 표현한다.

REST가 필요한 이유

1. 애플리케이션의 분리 및 통합 : Client와 Server의 완벽한 분리로 유지보수가 더 좋아지고, 사용자 인터페이스에 대한 관심을 데이터 저장에 대한 관심으로부터 분리함으로써 클라이언트 이식성과 서버의 규모확장성을 개선한다.
2. 다양한 클라이언트의 등장 : 최근의 서버 프로그램은 다양한 브라우저와 안드로이드 폰, 아이폰과 같은 모바일 디바이스에서도 통신을 할 수 있어야 한다.

REST 구성

1. **자원 (Resource)**
   URI로 표현되며, 해당 소프트웨어가 관리하는 모든 것
   ex) 문서, 그림, 데이터, 해당 소프트웨어 자체 등

2. **행위 (Verb)**
   HTTP 프로토콜 Method를 사용한다.
   (GET - 조회, POST - 생성, PUT - 전체 수정, PATCH - 일부 수정, DELETE - 삭제)

3. **표현 (Representation of Resource)**
   Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server가 보내는 응답.
   JSON, **XML**, **TEXT**, **RSS** 등 여러 형태의 Representation으로 나타내어진다.

   REST API 규칙 : 자원은 명사형으로 표현하고 행위는 동사형으로 표현한다.
   e.g) GET /members/10
   e.g) DELETE /members/1

REST의 특징

1. **Client - Server**
    - 자원이 있는 쪽이 Server, 자원을 요청하는 쪽이 Client가 되고 서로간의 의존성이 줄어든다.
    - Server : API를 제공하고 비즈니스 로직 처리 및 저장을 책임진다.
    - Client : 사용자 인증이나 Context (세션, 로그인 정보) 등을 직접 관리한다.

2. **Stateless (무상태)**
    - 세션과 쿠키같은 Client의 context 정보(상태)를 Server에 저장하지 않고 들어오는 요청만을 단순히 처리하면 되기에 구현이 단순해진다.

3. **Cacheable (캐시 처리 가능)**
    - 웹 표준 HTTP 프로토콜을 그대로 사용하므로 웹에서 사용하는 기존의 인프라를 그대로 활용할 수 있고 그 중 하나가 캐싱 기능이다. 캐싱은 Last-Modified 헤더와 ETag 헤더를 통해 검사하여 적용된다.
        - Last-Modified : 브라우저가 서버로 요청한 파일의 최종 수정 시간을 알려주는 헤더. 이미지/CSS/JS와 같은 정적파일들은 아파치에서 자동적으로 Last-Modified, If-Modified-Since헤더를 붙여준다. php와 같은 동적 파일들에는 로직상에서 헤더를 붙여주면 된다.
          e.g) `Last-Modified: Mon, 03 Jan 2011 17:45:57 GMT`
        - ETag (Entity Tag) : Last-Modified 방식과 거의 동일한데, 시간 대신 Hash 값을 사용한다.
          e.g) `ETag: "15f0fff99ed5aae4edffdd6496d7131f"`

4. **Uniform Interface (균일 인터페이스)**
    - URI로 지정한 Resource에 대한 조작을 통일되고 한정적인 인터페이스로 수행한다.
    - HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하여 특정 언어나 기술에 종속되지 않는다.

5. **Layered System (계층 구조)**
    - REST 서버는 다중 계층으로 구성될 수 있으며, 순수 비즈니스 로직, 보안, 로드 밸런싱, 암호화 계층을 추가해 구조상의 유연성을 둘 수 있다.
    - Proxy, 게이트웨이 같은 네트워크 기반의 중간매체를 사용할 수 있다.

6. **Code-On-Demand** (Optional)
    - Server로부터 스크립트를 받아 Client에서 실행한다.

REST 리소스 원형

1. **도큐먼트** : 객체나 데이터베이스의 row 단위의 데이터 [**단수 명사 사용**]
2. **컬렉션** : 도큐먼트의 집합. 관리 주체는 서버 [**복수 명사 사용**]
3. **스토어** : 도큐먼트의 집합. 관리 주체는 클라이언트 [**복수 명사 사용**]
4. **컨트롤러** : 컬렉션, 스토어의 메서드의 기능 [**동사나 동사구 사용**]

REST API 설계 규칙

1. 슬래시(/)는 계층 관계를 나타내는 데 사용한다.
2. URI 마지막 문자로 슬래시를 포함하지 않는다.
3. 긴 문자는 하이픈(-)을 사용하여 가독성을 높인다.
4. 밑줄(_)은 사용하지 않는다.
5. 동사가 아닌 명사를, 대문자가 아닌 소문자를 사용한다.
6. 확장자를 포함시키지 않는다.

   e.g) http://restapi.example.com/sports/soccer/players/player-of-the-year

응답 상태코드

1. **1XX** : 전송 프로토콜 수준의 정보 교환
2. **2XX** : 클라이언트 요청이 성공적으로 수행됨
3. **3XX** : 클라이언트가 요청을 완료하기 위해선 추가적인 행동을 취해야 함
4. **4XX** : 클라이언트의 잘못된 요청
5. **5XX** : 서버쪽 오류