Rest API

Status	Not started
Assign	웹

API (Application Programming Interface)

1. 정의

- API는 애플리케이션 소프트웨어를 구축하고 통합하기 위한 정의 및 프로토콜 세트
- 데이터와 기능의 집합을 제공하여 컴퓨터 프로그램간 상호작용을 촉진하며, 서로 정보를 교환가능 하도록 하는 것

2. 이해하기

```
배달주문의 과정을 생각해보자

1. 음식주문배달을 받는다. (client는 서버에 요청한다.)
2. 배달 할 집주소를 얻는다. (요청받은 서버는 응답을 줄 cliet 주소를 얻는다.)
3. 집주소에 맞게 음식을 전달한다. (데이터를 body에 넣어 client에 전달한다.)
4. 주문자는 음식을 전달받는다. (데이터을 받은 client는 화면에 데이터를 보여준다.)

이렇게,
API는 URI라는 서버주소와 클라이언트주소를 알고 http통신을 통해 서로 데이터를 주고받는다.
```

3. REST(Representation State Transfer)란?

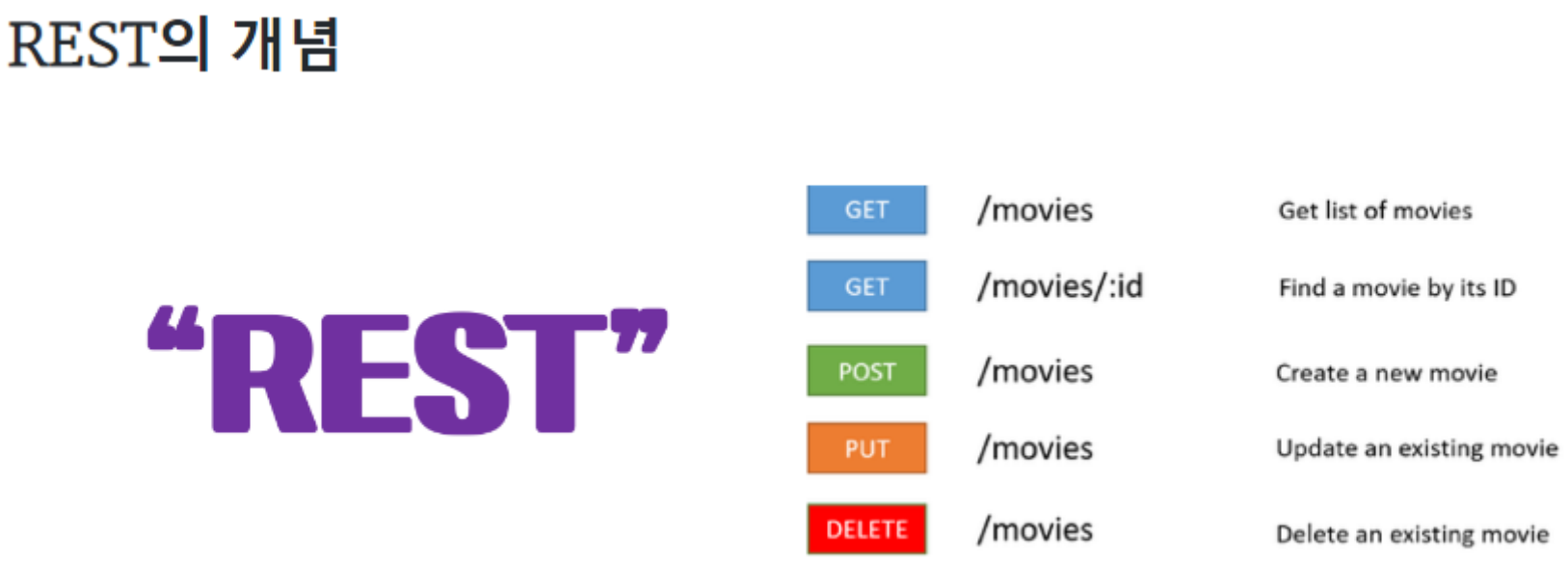
위키백과

월드 와이드 웹(World Wide Web, www)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다.

REST는 네트워크 아키텍처 원리의 모음이다. 여기서 '네트워크 아키텍처 원리'란 자원을 정의하고 자원에 대한 주소를 지정하는 방법 전반을 일컫는다.

간단한 의미로는, 웹 상의 자료를 HTTP위에서 SOAP이나 쿠키를 통한 세션 트랙킹 같은 별도의 전송 계층 없이 전송하기 위한 아주 간단한 인터페이스를 말한다. 이 두 가지의 의미는 겹치는 부분과 충돌되는 부분이 있다.

필딩의 **REST**아키텍처 형식을 따르면 HTTP나 WWW가 아닌 아주 커다란 소프트웨어 시스템을 설계하는 것도 가능하다. 또한, 리모트 프로시콜 대신에 간단한 XML과 HTTP인터페이스를 이용해 설계하는 것도 가능하다.



HTTP URI('https://domain.com:80/login')를 통해서 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미

```
GET https://example.com/search?name=foo HTTP/1.0
GET요청 : 데이터를 요청하는 메서드
서버API(https://example.com) : 서버 주소
params, query : 보다 명확한 구별가능한 자원을 명시
```

REST 특징

1. Server - Client 구조

- 서버, 클라이언트 역할이 구분되어 개발해야할 내용이 명확해지고 서로간 의존성이 줄어듬

> 클라이언트/서버 구조 : 아키텍처를 단순화시키고 작은 단위로 분리(decouple)함으로써 클라이언트-서버의 각 파트가 독립적으로 개선될 수 있도록 해준다..

2. Stateless - 무상태성

- 작업을 위한 상태 정보를 따로 저장하지 않음
- 들어오는 요청만 처리하면되서 서비스의 자유도가 올라감, 구현이 단순해짐

> **무상태(Stateless)**: 각 요청 간 클라이언트의 콘텍스트가 서버에 저장되어서는 안 된다

3. Cacheable - 캐시 처리 가능

- HTTP 웹표준을 그대로 사용하기 때문에 웹에서 사용하는 기존 인프라를 그대로 사용가능함, 따라서 HTTP의 캐싱 기능이 적용가능

> 캐시 처리 가능(Cacheable): WWW에서와 같이 클라이언트는 응답을 캐싱할 수 있어야 한다.
>
> 잘 관리되는 캐싱은 클라이언트-서버 간 상호작용을 부분적으로 또는 완전하게 제거하여 scalability와 성능을 향상시킨다.

4. Self-descriptiveness - 자체 표현 구조

- REST API 요청만 보고도 쉽게 이해할수 있는 자체 표현구조

5. Uniform Interface

- URI로 지정한 리소스에 대한 조작을 통일되고 한정적인 인터페이스로 수행하는 아키텍처 스타일을 말함

> 인터페이스 일관성 : 일관적인 인터페이스로 분리되어야 한다

6. 계층형 구조

- REST 서버는 다중 계층으로 구성될수 있으며 보안, 로드밸런싱, 암호화 계층을 추가해 구조상의 유연성을 둘 수 있고
- PROXY, 게이트웨이 같은 네트워크 기반의 중간 메체를 사용할 수 있음

> 계층화(Layered System): 클라이언트는 보통 대상 서버에 직접 연결되었는지, 또는 중간 서버를 통해 연결되었는지를 알 수 없다. 중간 서버는 로드 밸런싱 기능이나 공유 캐시 기능을 제공함으로써 시스템 규모 확장성을 향상시키는 데 유용하다.

설계 기본 규칙

- URI는 정보의 자원을 표현해야 한다.
- resource는 동사보다는 명사를, 대문자보다는 소문자를 사용한다.
- resource의 도큐먼트 이름으로는 단수 명사를 사용해야 한다.

- resource의 컬렉션 이름으로는 복수 명사를 사용해야 한다.
- resource의 스토어 이름으로는 복수 명사를 사용해야 한다.>GET /Member/1 -> GET /members/1
- 자원에 대한 행위는 HTTP Method(GET, PUT, POST, DELETE 등)로 표현한다.
- URI에 HTTP Method가 들어가면 안된다.> GET /members/delete/1 -> DELETE /members/1
- URI에 행위에 대한 동사 표현이 들어가면 안된다.(즉, CRUD 기능을 나타내는 것은 URI에 사용하지 않는다.)> (x)GET /members/show/1 -> (o)GET /members/1
- 경로 부분 중 변하는 부분은 유일한 값으로 대체한다.(즉, :id는 하나의 특정 resource를 나타내는 고유값이다.)> student를 생성하는 route: POST /students> id=12인 student를 삭제하는 route: DELETE /students/12

REST API 설계 규칙

- 슬래시 구분자(/)는 계층 관계를 나타내는데 사용한다.> houses/apartments
- URI 마지막 문자로 슬래시(/)를 포함하지 않는다.
- URI에 포함되는 모든 글자는 리소스의 유일한 식별자로 사용되어야 하며 URI가 다르다는 것은 리소스가 다르다는 것이고, 역으로 리소스가 다르면 URI도 달라져야 한다.
- REST API는 분명한 URI를 만들어 통신을 해야 하기 때문에 혼동을 주지 않도록 URI 경로의 마지막에는 슬래시(/)를 사용하지 않는다.> houses/apartments/ (X)
- 하이픈(-)은 URI 가독성을 높이는데 사용> 불가피하게 긴 URI경로를 사용하게 된다면 하이픈을 사용해 가독성을 높인다.
- 밑줄(_)은 URI에 사용하지 않는다.> 밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 하므로 가독성을 위해 밑줄은 사용하지 않는다.
- URI 경로에는 소문자가 적합하다. URI 경로에 대문자 사용은 피하도록 한다.> RFC 3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문
- 파일확장자는 URI에 포함하지 않는다.> REST API에서는 메시지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않으며, Accept header를 사용한다.> members/soccer/345/photo.jpg (X)> GET / members/soccer/345/photo HTTP/1.1 Host: restapi.example.com Accept: image/jpg (O)
- 리소스 간에는 연관 관계가 있는 경우> /리소스명/리소스 ID/관계가 있는 다른 리소스명> GET : /users/{userid}/devices (일반적으로 소유 ‘has’의 관계를 표현할 때)