

- REST의 정의
 - “Representational State Transfer”의 약자
 - 자원을 이름(자원의 표현)으로 구분하여 해당 자원의 상태(정보)를 주고 받는 모든 것을 의미한다.
 - 즉, 자원(resource)의 표현(representation)에 의한 상태 전달
 - a. 자원(resource)의 표현(representation)
 - 자원: 해당 소프트웨어가 관리하는 모든 것
 - -> Ex) 문서, 그림, 데이터, 해당 소프트웨어 자체 등
 - 자원의 표현: 그 자원을 표현하기 위한 언어를 사용
JSON, XML...
 - -> Ex) DB의 학생 정보가 자원일 때, ‘students’를 자원의 표현으로 정한다.
 - b. 상태(정보) 전달
 - 데이터가 요청되어지는 시점에서 자원의 상태(정보)를 전달한다.
 - JSON 혹은 XML을 통해 데이터를 주고 받는 것이 일반적이다.
 - 월드 와이드 웹(www)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 개발 아키텍처의 한 형식
 - REST는 기본적으로 웹의 기존 기술과 HTTP 프로토콜을 그대로 활용하기 때문에 웹의 장점을 최대한 활용할 수 있는 아키텍처 스타일이다.
 - REST는 네트워크 상에서 Client와 Server 사이의 통신 방식 중 하나이다.
- REST의 구체적인 개념
 - ① HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미한다.
 - 즉, REST는 자원 기반의 구조(ROA, Resource Oriented Architecture) 설계의 중심에 Resource가 있고 HTTP Method를 통해 Resource를 처리하도록 설계된 아키텍쳐를 의미한다.
 - 웹 사이트의 이미지, 텍스트, DB 내용 등의 모든 자원에 고유한 ID인 HTTP URI를 부여한다.
 - ② CRUD Operation
 - Create : 생성(POST)
 - Read : 조회(GET)
 - Update : 수정(PUT)
 - Delete : 삭제(DELETE)
 - HEAD: header 정보 조회(HEAD)

REST의 장단점

- 장점
 - HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구축할 필요가 없다.
 - HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해준다.
 - HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.
 - Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.
 - REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있다.
 - 여러가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.
 - 서버와 클라이언트의 역할을 명확하게 분리한다.
- 단점
 - 표준이 존재하지 않는다.
 - 사용할 수 있는 메소드가 4가지 밖에 없다.
 - HTTP Method 형태가 제한적이다.
 - 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 값이 웬지 더 어렵게 느껴진다.
 - 구형 브라우저가 아직 제대로 지원해주지 못하는 부분이 존재한다.
 - PUT, DELETE를 사용하지 못하는 점
 - pushState를 지원하지 않는 점

REST가 필요한 이유

- ‘애플리케이션 분리 및 통합’
- ‘다양한 클라이언트의 등장’
- 최근의 서버 프로그램은 다양한 브라우저와 안드로이폰, 아이폰과 같은 모바일 디바이스에서도 통신을 할 수 있어야 한다.
- 이러한 멀티 플랫폼에 대한 지원을 위해 서비스 자원에 대한 아키텍처를 세우고 이용하는 방법을 모색한 결과, REST에 관심을 가지게 되었다.

REST 구성 요소

1. 자원(Resource): URI
 - 모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재한다.
 - 자원을 구별하는 ID는 '/groups/:group_id'와 같은 HTTP URI다.
 - Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)에 대한 조작을 Server에 요청한다.
2. 행위(Verb): HTTP Method
 - HTTP 프로토콜의 Method를 사용한다.
 - HTTP 프로토콜은 GET, POST, PUT, DELETE 와 같은 메서드를 제공한다.
3. 표현(Representation of Resource)
 - Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보낸다.
 - REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타나 어질 수 있다.
 • JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적이다.

REST 특징

1. Server-Client(서버-클라이언트 구조)

- 자원이 있는 쪽이 Server, 자원을 요청하는 쪽이 Client가 된다.
 - REST Server: API를 제공하고 비즈니스 로직 처리 및 저장을 책임진다.
 - Client: 사용자 인증이나 context(세션, 로그인 정보) 등을 직접 관리하고 책임진다.
- 서로 간 의존성이 줄어든다.

2. Stateless(무상태)

- HTTP 프로토콜은 Stateless Protocol이므로 REST 역시 무상태성을 갖는다.
- Client의 context를 Server에 저장하지 않는다.
 - 즉, 세션과 쿠키와 같은 context 정보를 신경쓰지 않아도 되므로 구현이 단순해진다.
- Server는 각각의 요청을 완전히 별개의 것으로 인식하고 처리한다.
 - 각 API 서버는 Client의 요청만을 단순 처리한다.
 - 즉, 이전 요청이 다음 요청의 처리에 연관되어서는 안된다.
 - 물론 이전 요청이 DB를 수정하여 DB에 의해 바뀌는 것은 허용한다.
 - Server의 처리 방식에 일관성을 부여하고 부담이 줄어들며, 서비스의 자유도가 높아진다.

3. Cacheable(캐시 처리 가능)

- 웹 표준 HTTP 프로토콜을 그대로 사용하므로 웹에서 사용하는 기존의 인프라를 그대로 활용할 수 있다.
 - 즉, HTTP가 가진 가장 강력한 특징 중 하나인 캐싱 기능을 적용할 수 있다.
 - HTTP 프로토콜 표준에서 사용하는 Last-Modified 태그나 E-Tag를 이용하면 캐싱 구현이 가능하다.
- 대량의 요청을 효율적으로 처리하기 위해 캐시가 요구된다.
- 캐시 사용을 통해 응답시간이 빨라지고 REST Server 트랜잭션이 발생하지 않기 때문에 전체 응답시간, 성능, 서버의 자원 이용률을 향상시킬 수 있다.

4. Layered System(계층화)

- Client는 REST API Server만 호출한다.
- REST Server는 다중 계층으로 구성될 수 있다.
 - API Server는 순수 비즈니스 로직을 수행하고 그 앞단에 보안, 로드밸런싱, 암호화, 사용자 인증 등을 추가하여 구조상의 유연성을 줄 수 있다.
 - 또한 로드밸런싱, 공유 캐시 등을 통해 확장성과 보안성을 향상시킬 수 있다.
- PROXY, 게이트웨이 같은 네트워크 기반의 중간 매체를 사용할 수 있다.

5. Code-On-Demand(optional)

- Server로부터 스크립트를 받아서 Client에서 실행한다.
- 반드시 충족할 필요는 없다.

6. Uniform Interface(인터페이스 일관성)

- URI로 지정한 Resource에 대한 조작을 통일되고 한정적인 인터페이스로 수행한다.
- HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.
 - 특정 언어나 기술에 종속되지 않는다.

REST API의 개념

{ REST API }

Representational State Transfer API

REST API란

- API(Application Programming Interface)란
 - 데이터와 기능의 집합을 제공하여 컴퓨터 프로그램간 상호작용을 촉진하며, 서로 정보를 교환가능 하도록 하는 것
- REST API의 정의
 - ~~• REST 기반으로 서비스 API를 구현한 것~~
 - 최근 OpenAPI(누구나 사용할 수 있도록 공개된 API: 구글 맵, 공공 데이터 등), 마이크로 서비스(하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처) 등을 제공하는 업체 대부분은 REST API를 제공한다.

REST API의 특징

- 사내 시스템들도 REST 기반으로 시스템을 분산해 확장성과 재사용성을 높여 유지보수 및 운용을 편리하게 할 수 있다.
- REST는 HTTP 표준을 기반으로 구현하므로, HTTP를 지원하는 프로그램 언어로 클라이언트, 서버를 구현할 수 있다.
- 즉, REST API를 제작하면 델파이 클라이언트 뿐 아니라, 자바, C#, 웹 등을 이용해 클라이언트를 제작할 수 있다.

REST API 설계 기본 규칙

참고 리소스 원형

- **도큐먼트** : 객체 인스턴스나 데이터베이스 레코드와 유사한 개념 (단수)
- **컬렉션** : 서버에서 관리하는 "디렉터리라는" 리소스 (복수)
- **스토어** : 클라이언트에서 관리하는 리소스 저장소

1. URI는 정보의 자원을 표현해야 한다.

- i. resource는(동사보다는)명사를,(대문자보다는)소문자를 사용한다.
- ii. resource의 도큐먼트 이름으로는 단수 명사를 사용해야 한다.
- iii. resource의 컬렉션 이름으로는 복수 명사를 사용해야 한다.
- iv. resource의 스토어 이름으로는 복수 명사를 사용해야 한다.

- Ex) GET /Member/1 -> GET /members/1

2. 자원에 대한 행위는 HTTP Method(GET, PUT, POST, DELETE 등)로 표현한다.

- i. URI에 HTTP Method가 들어가면 안된다.
 - Ex) GET /members/delete/1 -> DELETE /members/1
- ii. URI에 행위에 대한 동사 표현이 들어가면 안된다.(즉, CRUD 기능을 나타내는 것은 URI에 사용하지 않는다.)
 - Ex) GET /members/show/1 -> GET /members/1
 - Ex) GET /members/insert/2 -> POST /members/2
- iii. 경로 부분 중 변하는 부분은 유일한 값으로 대체한다.(즉, :id는 하나의 특정 resource를 나타내는 고유 값이다.)
 - Ex) student를 생성하는 route: POST /students
 - Ex) id=12인 student를 삭제하는 route: DELETE /students/12

REST API 설계 규칙

1. 슬래시 구분자(/)는 계층 관계를 나타내는데 사용한다.
 - Ex) `http://restapi.example.com/houses/apartments`
2. 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.
 - Ex) `http://restapi.example.com/houses/apartments/` (X)
3. 하이픈(-)은 URI 가독성을 높이는데 사용
 - 불가피하게 긴 URI경로를 사용하게 된다면 하이픈을 사용해 가독성을 높인다.
4. 밑줄(_)은 URI에 사용하지 않는다.
 - 밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 하므로 가독성을 위해 밑줄은 사용하지 않는다.
5. URI 경로에는 소문자가 적합하다.
 - URI 경로에 대문자 사용은 피하도록 한다.
 - RFC 3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문
6. 파일확장자는 URI에 포함하지 않는다.
 - REST API에서는 메시지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않는다.
 - Accept header를 사용한다.
 - Ex) `http://restapi.example.com/members/soccer/345/photo.jpg` (X)
 - Ex) `GET / members/soccer/345/photo HTTP/1.1 Host: restapi.example.com Accept: image/jpg` (O)
7. 리소스 간에는 연관 관계가 있는 경우
 - /리소스명/리소스 ID/관계가 있는 다른 리소스명
 - Ex) `GET : /users/{userid}/devices` (일반적으로 소유 'has'의 관계를 표현할 때)

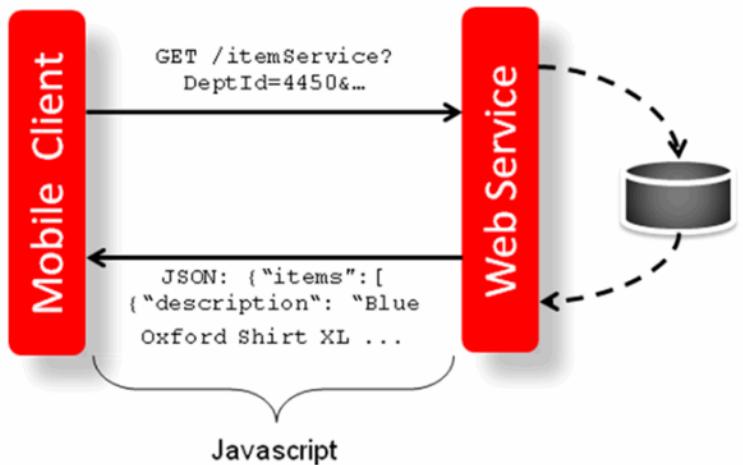
REST API 설계 예시

CRUD	HTTP verbs	Route
resource들의 목록을 표시	GET	/resource
resource 하나의 내용을 표시	GET	/resource/:id
resource를 생성	POST	/resource
resource를 수정	PUT	/resource/:id
resource를 삭제	DELETE	/resource/:id

참고 응답상태코드

- 1XX : 전송 프로토콜 수준의 정보 교환
- 2XX : 클라이언트 요청이 성공적으로 수행됨
- 3XX : 클라이언트는 요청을 완료하기 위해 추가적인 행동을 취해야 함
- 4XX : 클라이언트의 잘못된 요청
- 5XX : 서버쪽 오류로 인한 상태코드

RESTful의 개념



“RESTful”

RESTful이란

- RESTful은 일반적으로 REST라는 아키텍처를 구현하는 웹 서비스를 나타내기 위해 사용되는 용어이다.
~~• ‘REST API’를 제공하는 웹 서비스를 ‘RESTful’하다고 할 수 있다.~~
- RESTful은 REST를 REST답게 쓰기 위한 방법으로, 누군가가 공식적으로 발표한 것이 아니다.
 - 즉, REST 원리를 따르는 시스템은 RESTful이란 용어로 지칭된다.

RESTful의 목적

- 이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것
- RESTful한 API를 구현하는 근본적인 목적이 성능 향상에 있는 것이 아니라 일관적인 컨벤션을 통한 API의 이해도 및 호환성을 높이는 것이 주 동기이니, 성능이 중요한 상황에서는 굳이 RESTful한 API를 구현할 필요는 없다.

RESTful 하지 못한 경우

- Ex1) CRUD 기능을 모두 POST로만 처리하는 API
- Ex2) route에 resource, id 외의 정보가 들어가는 경우(/students/updateName)