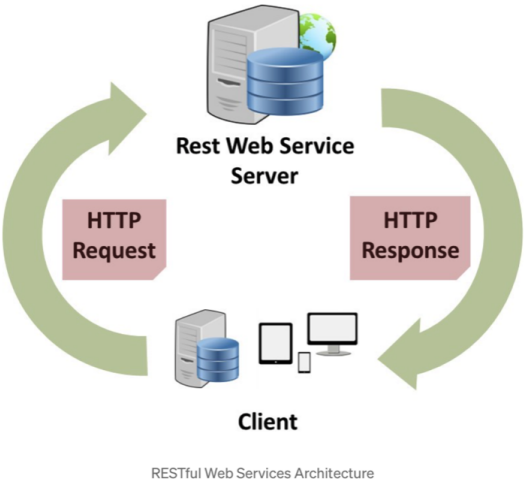
RESTFUL

REST API

REST API란?

- REST API에서 REST는 Representational State Transfer의 약자로 소프트웨어 프로그램 아키텍처의 한 형식이다.
- 즉, 자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것을 의미한다.
- 월드 와이드 웹(WWW)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 개발 아키텍처의 한 형식이다.
- REST는 기본적으로 웹의 기존 기술과 HTTP프로토콜을 그대로 활용하기 때문에 웹의 장점을 최대한 활용할 수 있는 아키텍처 스타일이다.



RESTful Web Services Architecture

REST의 구체적인 개념

- HTTP URI를 통해 자원을 명시하고, HTTP Method(POST, GET, PUT / PATCH, 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 아키텍처를 의미한다.
- **웹의 모든 자원에 고유한 ID인 HTTP URI를 부여한다.**

REST API 탄생 배경

- REST API의 등장은 2000년도에 HTTP의 주요 저자 중 한 사람인 로이 필딩이 그 당시 웹 설계의 우수성에 비해 제대로 사용되어지지 못하는 모습에 안타까워하며 **웹의 장점을 최대한 활용할 수 있는 아키텍쳐로써 REST를 발표했다.**
- 최근의 서비스 / 어플리케이션의 개발 흐름은 멀티 플랫폼, 멀티 디바이스 시대로 넘어와 있다. 단순히 하나의 브라우저만 지원하면 되었던 이전과는 달리, 최근의 서버 프로그램은 여러 웹 브라우저는 물론이며, 아이폰, 안드로이드 애플리케이션과의 통신에 대응할 수 있어야 한다.
- 따라서 플랫폼에 맞추어 새로운 서버를 만드는 수고를 들이지 않기 위해 범용적으로 사용성을 보장하는 서버 디자인이 필요하게 되었습니다.

우리는 왜 RESTful APIs를 만드는 것일까?

- RESTful APIs 개발하는 가장 큰 이유는 Client Side를 정형화된 플랫폼이 아닌 모바일, PC, 어플리케이션 등 플랫폼에 제약을 두지 않는 것을 목표로 했기 때문이다.
- 즉, 2010년 이전만 해도 Server Side에서 데이터를 전달해주는 Client 프로그램의 대상은 PC 브라우저로 그 대상이 명확했다. 그렇다 보니 그냥 JSP ASP PHP 등 이용한 웹페이지를 구성하고 작업을 진행하면 됐다.
- 하지만 스마트 기기들이 등장하면서 TV, 스마트 폰, 태블릿 등 Client 프로그램이 다양화되고 그에 맞춰 Server를 일일이 만든다는 것이 꽤 비효율적인 일이 되어버렸다.
- 이런 과정에서 개발자들은 Client Side를 전혀 고려하지 않고 메세지 기반, XML, JSON과 같은 Client에서 바로 객체로 치환 가능한 형태의 데이터 통신을 지향하게 되면서 Server와 Client의 역할을 분리하게 되었다.

→ 이런 변화를 겪으면서 필요해진 것은 **HTTP 표준 규약**을 지키면서 API를 만드는 것이다.

유의사항

- 이 REST API를 개발하다보면 HTTP의 Response 규약을 지키지 않고 본인들이 만들어낸 JSON convention으로 응답하는 것을 많이 볼 수 있는데 그것은 옳지 않은 개발 방향이다.
- 왜냐하면 Client Side가 정형화 되어있지 않은 환경에서 개발 속도를 저하하는 가장 큰 이유는 표준을 지키지 않았기 때문이다.

REST의 구성

- 자원 -URL
- 행위 - Http Method
- 표현 - Representation

1. **자원 - URL**
    a. 모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재한다.
    b. 자원을 구별하는 ID는 /order/order_id/1 과 같은 **HTTP URI**이다.
2. **행위 - HTTP Method**
    a. HTTP 프로토콜의 Method를 사용한다.
    b. HTTP 프로토콜은 GET, POST, PUT, DELETE와 같은 메소드를 제공한다.
3. **표현**
    a. Client가 자원의 상태에 대한 조작을 요청하면 Server는 이에 적절한 응답을 보낸다.
    b. REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타낼 수 있다.
    c. 현재는 JSON으로 주고 받는 것이 대부분이다.

[참고]

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성한다.
GET	GET을 통해 해당 리소스를 조회한다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정볼르 가져온다.
PUT	PUT을 통해 해당 리소스를 수정한다.
DELETE	DELETE를 통해 리소스를 삭제한다.

REST특징

1. **클라이언트 / 서버 구조**
    a. 클라이언트는 유저와 관련된 처리를, 서버는 REST API를 제공함으로써 각각의 역할이 확실하게 구분되고 일괄적인 인터페이스로 분리되어 작동할 수 있게 한다
    b. REST Server : API를 제공하고 비즈니스 로직 처리 및 저장을 책임진다.
    c. Client : 사용자 인증이나 context (세션, 로그인 정보) 등을 직접 관리하고 책임진다.
    d. 서로 간 의존성이 줄어든다.
2. **무상태성 (Stateless)**
    a. REST는 HTTP의 특성을 이용하기 때문에 무상태성을 갖는다.
    b. 즉 서버에서 어떤 작업을 하기 위해 상태정보를 기억할 필요가 없고 들어온 요청에 대해 처리만 해주면 되기 때문에 구현이 쉽고 단순해진다.
3. **캐시 처리 기능 (Cacheable)**

    a. HTTP라는 기존 웹표준을 사용하는 REST의 특징 덕분에 기본 웹에서 사용하는 인프라를 그대로 사용 가능하다.
    b. 대량의 요청을 효율적으로 처리하기 위해 캐시가 요구된다.
    c. 캐시 사용을 통해 응답시간이 빨라지고 REST Server 트랜잭션이 발생하지 않기 때문에 전체 응답시간, 성능, 서버의 자원 이용률을 향상 시킬 수 있다.
4. **자체 표현 구조 (Self -descriptiveness)**
    a. JSON을 이용한 메세지 포멧을 이용하여 직관적으로 이해할 수 있고 REST API 메세지만으로 그 요청이 어떤 행위를 하는지 알 수 있다.
5. **계층화 (Layered System)**
    a. 클라이언트와 서버가 분리되어 있기 때문에 중간에 프록시 서버, 암호화 계층 등 중간매체를 사용할 수 있어 자유도가 높다.
6. **유니폼 인터페이스 (Uniform)**
    a. Uniform Interface는 HTTP 표준에만 따른다면 모든 플랫폼에서 사용이 가능하며, URI로 지정한 리소스에 대한 조작을 가능하게 하는 아키텍쳐 스타일을 말한다.
    b. URI로 지정한 Resource에 대한 조작을 통일되고 한정적인 인터페이스로 수행한다.
    c. **즉, 특정 언어나 기술에 종속되지 않는다.**

중심 규칙

REST에서 가장 중요하며 기본적인 규칙은 아래 두 가지입니다.

- URI는 정보의 자원을 표현해야 한다.
- 자원에 대한 행위는 HTTP Method(GET, POST, PUT, DELETE 등)으로 표현한다.

세부 규칙

1. 슬래시 구분자 (/)는 계층 관계를 나타내는데 사용한다.
2. URI 마지막 문자로 슬래시 (/)를 포함하지 않는다.
    - 즉 URI에 포함되는 모든 글자는 리소스의 유일한 식별자로 사용되어야 하고 URI가 다르다는 것은 리소스가 다르다는 것이다.
    - 역으로 리소스가 다르면 URI도 달라져야 한다.
3. 하이픈 (-)은 URI 가독성을 높이는 데 사용한다.
4. 밑줄 (_)은 URI에 사용하지 않는다.
5. URI 경로에는 소문자가 적합하다.
    a. URI 경로에 대문자 사용은 피하도록 한다.
6. 파일확장자는 URI에 포함하지 않는다.
    a. REST API에서는 메세지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않는다.
    b. 대신 Accept Header를 사용한다.

c. ex) GET: http://restapi.exam.com/orders/2/Accept: image/jpg

7. 리소스 간에 연관 관계가 있는 경우
   a. /리소스명/리소스ID/관계가 있는 다른 리소스명
   b. ex) GET: /users/2/orders (일반적으로 소유의 관계를 표현할 때 사용)

설계 목표

REST의 핵심 설계 목표는 다음과 같다

1. 컴포넌트들간의 유연한(쉽게 확장 가능한) 상호 연동성 확보
2. 범용 인터페이스
3. 각 컴포넌트들의 독립적인 배포
4. 지연 감소, 보안 강화, 레거시 시스템을 **encapsulation**하는 중간 컴포넌트로의 역할
   - encapsulation : 데이터가 헤더에 추가되는 과정 즉, OS Lv7 → Lv1로 내려 가는 과정을 뜻하며 간단하게 말하면 PC에서 다른 PC로 데이터를 전송할 때 데이터를 패키지화하는 과정을 말한다.

1) 상호연동성 확보

- 상호연동성은 "서로 상이한 컴포넌트"들을 쉽게 연결할 수 있는 성질을 의미한다, 상호연동성은 두 개 이상의 컴포넌트들을 결합함으로써, 작업을 더 효율적으로 수행하도록 하는데 목적이 있다.

2) 범용 인터페이스

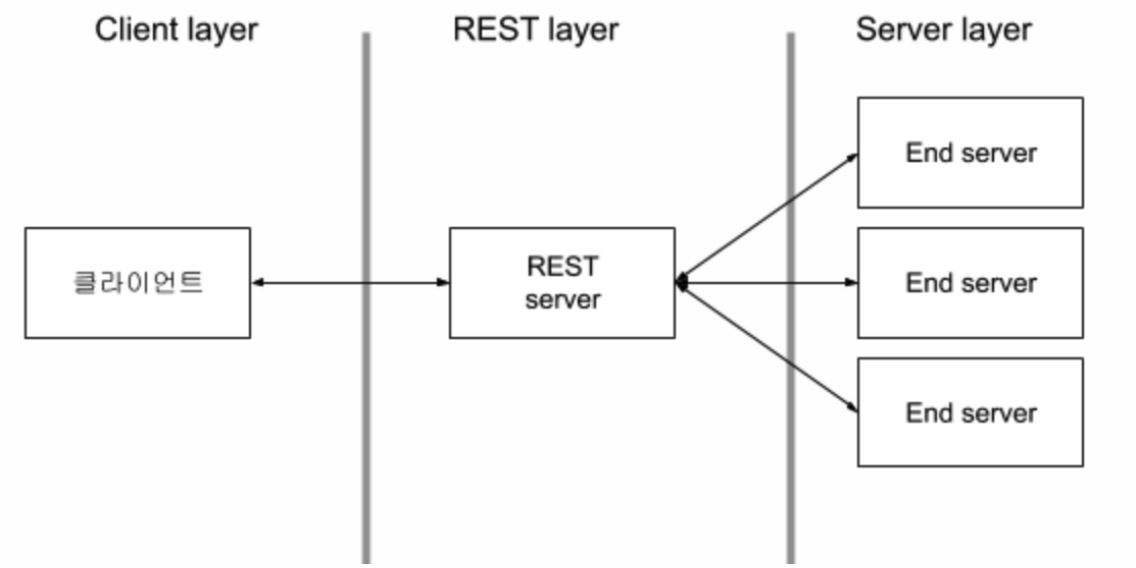
- 상호연동성과 비슷한 개념으로 REST 모델을 위한 HTTP와 URI는 표준으로 어디서든지 사용가능한 범용 인터페이스를 제공한다.
- **따라서 개발자는 비즈니스 로직만 고민하면 된다.**

3) 각 컴포넌트들의 독립적인 배포

- 각 컴포넌트들과의 독립적인 배포의 의미는 다른 컴포넌트들과 독립적으로 개발을 할 수 있다는 것을 의미한다.
- 규격에 맞추어 개발이 되었다면 다른 컴포넌트가 추가되어도 연동에 걱정할 것이 없다.

4) 컴포넌트를 중계하는 역할

- 클라이언트는 엔드 서버에 직접 연결할 필요 없이 서비스를 이용할 수 있다. 그 이유는 REST 서버가 클라이언트와 엔드 서버 중간에서 중계역할을 할 수 있기 때문이다.
- 이러한 중계 서버로 이용하면 로드 밸런싱, 공유 메모리 등을 이용하여 확장성/성능을 향상시킬 수 있으며 보안 정책을 적용하기도 용이하다.
  - 로드 밸런싱 : 부하 분산이라고도 불리며 컴퓨터 네트워크 기술의 일종으로 둘 혹은 셋 이상의 중앙처리장치 혹은 저장장치와 같은 컴퓨터 자원들에게 작업을 나누어 주는 것을 의미한다.



RESTful API

RESTful이란

- HTTP와 URI기반으로 자원에 접근할 수 있도록 제공하는 어플리케이션 개발 인터페이스이다.
- 기본적으로 개발자는 HTTP 메소드와 URI만으로 인터넷에 자료를 CRUD할 수 있다.
- ‘REST API’를 제공하는 웹 서비스를 ‘RESTful’하다고 할 수 있다.
- RESTful은 REST를 REST답게 쓰기 위한 방법으로, 누군가가 공식적으로 발표한 것은 아니다.

RESTful API 개발 원칙

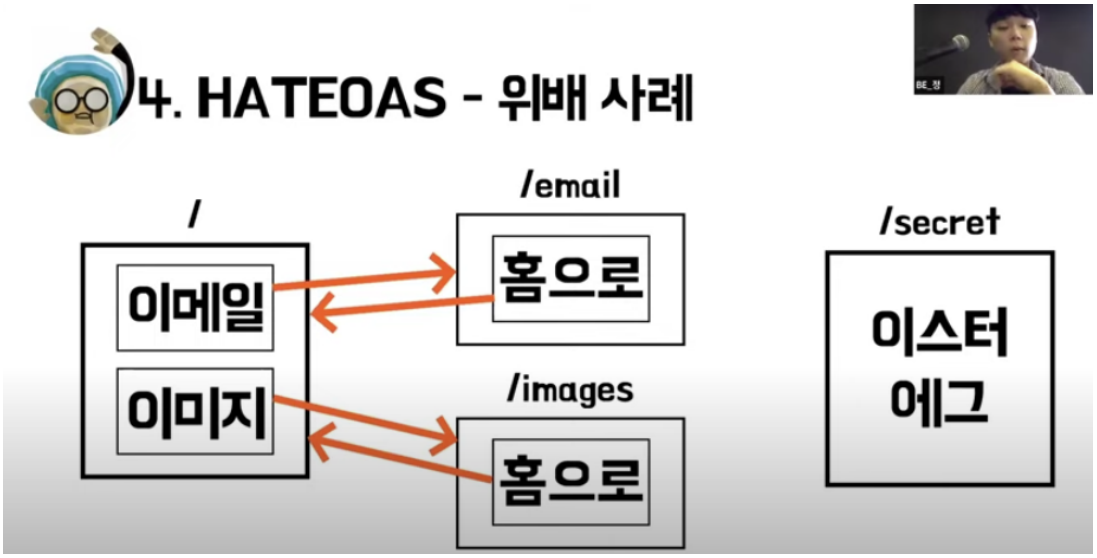
a. **자원을 식별할 수 있어야 한다.**
   - URL (Uniform Resource Locator) 만으로 내가 어떤 자원을 제어하려고 하는지 알 수 있어야 한다. 자원을 제어하기 위해서, 자원의 위치는 물론 자원의 종류까지 알 수 있어야 한다는 의미이다.
   - Server가 제공하는 정보는 JSON이나 XML형태로 HTTP body에 포함되어 전송시킨다.

b. **행위는 명시적이어야 한다.**
   - REST는 아키텍처 혹은 방법론과 비슷하다. 따라서 이런 방식을 사용해야 한다고 강제적이지 않다. 기존의 웹 서비스처럼, GET을 이용해서 UPDATE와 DELETE를 해도 된다.
   - 다만 REST 아키텍처에는 부합하지 않으므로 REST를 따른다고 할 수는 없다

c. **자기 서술적이어야 한다.**
   - 데이터에 대한 메타정보만 가지고도 어떤 종류의 데이터인지, 데이터를 위해서 어떤 어플리케이션을 실행해야 하는지를 알 수 있다.
   - **즉, 데이터 처리를 위한 정보를 얻기 위해서, 데이터 원본을 읽어야 한다면 자기 서술적이지 못하다.**

d. **HATEOAS (Hypermedia as the Engine of Application State)**

- 클라이언트 요청에 대해 응답을 할 때, 추가적인 정보를 제공하는 링크를 포함할 수 있어야 한다.
- REST는 독립적으로 컴포넌트들을 손쉽게 연결하기 위한 목적으로도 사용된다. 따라서 서로 다른 컴포넌트들을 유연하게 연결하기 위해서는, 느슨한 연결을 만들어줄 것이 필요하다.
- 이때 사용되는 것이 바로 **링크**이다. 서버는 클라이언트 응용 어플리케이션에 하이퍼 링크를 제공한다
- 클라이언트는 이 하이퍼 링크를 통해서 전체 네트워크와 연결되며 HATEOAS는 서버가 독립적으로 진화할 수 있도록 서버와 서버, 서버와 클라이언트를 분리할 수 있게 한다.



- 이렇게 숨겨진 페이지 있으면 안된다.

REST의 단점들

1. REST는 point-to-point 통신모델을 기본으로 한다. 따라서 서버와 클라이언트가 연결을 맺고 상호작용해야하는 어플리케이션의 개발에는 적당하지 않다.
2. REST는 URI, HTTP를 이용한 아키텍처링 방법에 대한 내용만을 담고 있다. 보안과 통신규약 정책같은 것은 전혀 다루지 않는다. 따라서 개발자는 통신과 정책에 대한 설계와 구현을 도맡아서 진행해야 한다.
3. HTTP에 상당히 의존적이다. REST는 설계 원리이기 때문에 HTTP와는 상관없이 다른 프로토콜에서도 구현될 수 있기는 하지만 자연스러운 개발이 힘들다. 다만 REST를 사용하는 이유가 대부분의 서비스가 웹으로 통합되는 상황이기에 큰 단점이 아니게 되었다.
4. CRUD 4가지 메소드만 제공한다. 대부분의 일들을 처리할 수 있지만, 4가지 메소드만으로 처리하기엔 모호한 표현이 있다.

References :

https://velog.io/@someday/RESTful-API-이란

https://www.youtube.com/watch?v=Nxi8Ur89Akw&t=15

https://www.youtube.com/watch?v=xWA1eTPSzDE

https://meetup.nhncloud.com/posts/92