RESTful API

REST(Representational State Transfer)는
'각 자원에 대해 자원의 상태에 대한 정보를 주고받는 개발 방식'
을 의미.

REST의 구성 요소
1) 자원(Resource) : URI 이용
2) 행위(Verb) : HTTP 메서드 이용
3) 표현(Representations) : 페이로드 이용.

e.g)
클라이언트 : 사용자(자원) 회원등록(행위)을 하고 싶어요.
아이디는 abc 비밀번호는 1234 로 설정하고 싶어요. (페이로드)

구성요소

-자원(Resource): URI

모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재.
자원을 구별하는 ID는 '/groups/:group_id'와 같은 HTTP URI .
Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)
에 대한 조작을 Server에 요청.

-행위(Verb): HTTP Method

HTTP 프로토콜의 Method를 사용.
HTTP 프로토콜은 GET, POST, PUT, DELETE 와 같은 메서드를 제공.

-표현(Representation of Resource)

Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보냄.
REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타내어 질 수 있고,
JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적.

- RESTful의 의미

REST란, **HTTP를 잘 사용하기 위한 아키텍쳐 스타일**.

REST는 URI와 HTTP 메소드를 사용해서 자원과 행위를 표현. REST의 원칙을 지키면서 API의 의미를 표현하고 쉽고, 파악하기 쉽게 하는 것을 Restful 하다고 함.

- 등장 이유

HTTP는 GET, POST, PUT, DELETE 등의 다양한 HTTP 메서드를 지원.
–실제로는 서버가 각 메서드의 기본 설명을 따르지 않더라도 프로그램을 개발할 수 있음.
–하지만 저마다 다른 방식으로 개발하면 문제가 될 수 있어서 **기준이 되는 아키텍처**가 필요

RESTful 만족하기 위한 조건

-client-server(클라이언트-서버 구조)
- stateless(상태없음)
- cache(캐시 처리 가능)
- uniform interface(인터페이스 일관성)
- layered system(계층화)
- code-on-demand(서버가 네트워크를 통해 클라이언트에 전달한 프로그램을 클라이언트가 실행할 수 있어야 함 *optional)

REST 특징

Server-Client(서버-클라이언트 구조)

자원이 있는 쪽이 Server, 자원을 요청하는 쪽이 Client.

REST Server: API를 제공하고 비즈니스 로직 처리 및 저장을 책임.

Client: 사용자 인증이나 context(세션, 로그인 정보) 등을 직접 관리하고 책임.

Stateless(무상태)

HTTP 프로토콜은 Stateless Protocol이므로 REST 역시 무상태성을 갖음

Client의 context를 Server에 저장하지 않음

->세션과 쿠키와 같은 context 정보를 신경쓰지 않아도 되므로 구현이 단순

Server는 각각의 요청을 완전히 별개의 것으로 인식하고 처리

각 API 서버는 Client의 요청만을 단순 처리.

즉, 이전 요청이 다음 요청의 처리에 연관되어서는 X

(이전 요청이 DB를 수정하여 DB에 의해 바뀌는 것은 허용)

Server의 처리 방식에 일관성을 부여하고 부담이 줄어들며, 서비스의 자유도가 높아짐.

Cacheable(캐시 처리 가능)

웹 표준 HTTP 프로토콜을 그대로 사용하므로 웹에서 사용하는 기존의 인프라를 그대로 활용할 수 있음.

대량의 요청을 효율적으로 처리하기 위해 캐시가 요구

-> 캐시 사용을 통해 응답시간이 빨라지고 REST Server 트랜잭션이 발생하지 않기 때문에 전체 응답시간, 성능, 서버의 자원 이용률을 향상시킬 수 있음

* 기본적으로 캐싱은 접근되어진 데이터들의 뭉치들을 저장하는 능력을 의미.

클라이언트는 서버를 위해 캐싱된 응답을 반환해주고, Http 프로토콜 표준에서 사용하는 Last-Modified태그나 E-Tag를 이용하면 캐싱 구현이 가능

Cacheable(캐시 처리 가능)

Last-Modified

응답 날짜 헤더는 언제 응답이 나타났는지 가리키는 데 반면에, Last-Modified 헤더는 지난 할당되어진 자원이 바뀔 때 가리킴-> Last-Modified value는 Date value보다 최근일 수가 없음!

ETag

ETag는 평생동안 리소스 상태를 확인하기 위해 서버랑 연관시키는 불투명한 문자 토큰. 리소스가 변할 때, ETag는 부응에 맞춰 바뀜.

eg) www.google.com 응답 본문이 'hi' , ETag 값이 11111일 때,
서버 컨텐츠가 바뀌어서 응답 본문이 'hello'라면, ETag 값도 12345로 바뀜.
-> 같은 get방식으로 요청을 하더라도 응답 본문이 달라지면 ETag 헤더 값이 변경

Layered System(계층화)

Client는 REST API Server만 호출.

REST Server는 다중 계층으로 구성될 수 있음.

API Server는 순수 비즈니스 로직을 수행하고 그 앞단에 보안, 로드밸런싱, 암호화, 사용자 인증 등을 추가하여 구조상의 유연성을 줌

또한 로드밸런싱, 공유 캐시 등을 통해 확장성과 보안성을 향상시킴

PROXY, 게이트웨이 같은 네트워크 기반의 중간 매체를 사용할 수 있음.

Code-On-Demand(optional)

Server로부터 스크립트를 받아서 Client에서 실행.

(반드시 충족할 필요X)

Uniform Interface(인터페이스 일관성)

URI로 지정한 Resource에 대한 조작을 통일되고 한정적인 인터페이스로 수행.

HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능.

(특정 언어나 기술에 종속X)

Uniform interface 란

URL로 지정된 리소스에 대한 조작을 통일하고 한정된 인터페이스로 수행하는 아키텍쳐 스타일

1) URL과 한정된 인터페이스

예전에는 param을 통해 해당 resource에 접속 가능

-> 하나의 param이라면 상관이 없지만, 상황에 따라서는 URI 주소가 길어짐

Rest API 특징 중에 하나가 URL을 쓰는 동시에 **한정된 인터페이스**로써 사용하면 굳이 주소를 길게 적을 필요 없이 한정된 자원으로도 해당 resource를 접근할 수 있는 장점을 가짐.

URI REST를 적용한 URL	http://elicetrack1234.com?auth=admin http://elicetrack1234.com/admin

2) 한정된 인터페이스의 의미

한정된 인터페이스란?

Resource	GET	PUT	POST	DELECT
http://elice1234.com/admin	관리자 페이지 조회	관리자 페이지 수정	관리자 페이지 생성	관리자 페이지 삭제

-GET, PUT, POST, DELECT 4가지 인터페이스로 한정지어서 해당하는 Resource를 접근 -> 다양한 방법으로 설명이 될 수 있는 Representation을 가짐

-리소스 자체를 전송하는 것이 아닌 리소스 표현을 전송. 이런 방식을 사용하게 되면 URL 주소 길이가 짧아 지고, 하나의 URL이 많은 Representation을 나타낼 수 있는 장점을 가짐.

RESTful Naming

- URI(Uniform Resource Identifier)
  - Uniform: 리소스 식별하는 통일된 방식
  - Resource: 자원, URI로 식별할 수 있는 모든 것(제한 없음)
  - Identifier: 다른 항목과 구분하는데 필요한 정보

- URI는 Resource(or Representation)만 식별함

eg) 미네랄을 캐라 -> 미네랄이 리소스, 회원을 등록하고 수정하는 기능 -> 회원이라는 개념 자체가 리소스

- 최근에는 Resource 대신 Representation 라는 표현을 사용하기 시작함

1. 리소스를 표현하기 위해 명사를 사용

RESTful URI이 가르키는 resource(or representation)는
수행되는 행위가 아니라 객체

[resource (or representation)의 네 가지 범주]

-> 항상 리소스가 어느 범주에 해당하는 지 확인하고, 그 범주에 맞는 네이밍 컨벤션을 일관되게 사용

- 문서(Document)

-단일 개념(파일 하나, 객체 인스턴스, 데이터베이스 row)

-단수 사용 (/device-management, /user-management

```
1  http://api.example.com/device-management/managed-devices/{device-id}
2  http://api.example.com/user-management/users/{id}
3  http://api.example.com/user-management/users/admin
```

- **컬렉션(Collection)**

-서버가 관리하는 리소스 디렉터리

-서버가 리소스의 URI를 생성하고 관리

-POST 기반 등록 예) 회원 관리 API

-복수 사용 (/users)

```
http://api.example.com/user-management/users
http://api.example.com/user-management/users/{id}
```

- **스토어(Store)**

-클라이언트가 관리하는 자원 저장소

-클라이언트가 리소스의 URI를 알고 관리

-PUT 기반 등록 예) 정적 컨텐츠 관리, 원격 파일 관리

-복수 사용 (/files)

```
http://api.example.com/files
http://api.example.com/files/new_file.txt
```

- 컨트롤 URI 혹은 컨트롤러(Controller)

-문서, 컬렉션, 스토어로 해결하기 어려운 추가 프로세스 실행

-동사를 직접 사용 (/checkout, /play 등)

-예) GET, POST만 사용할 수 있는 HTTP FORM의 경우 컨트롤 URI(동사로 된 리소스 경로)를 사용 ex) /members/delete, /members/new 등

```
1    http://api.example.com/cart-management/users/{id}/cart/checkout
2    http://api.example.com/song-management/users/{id}/playlist/play
```

2. 일관성이 핵심

일관된 resource (or representation) naming conventions과 URI 형식을 사용하면 모호함이 최소화되고 가독성과 지속성이 극대화, 일관성을 위해 다음과 같은 디자인 힌트를 구현할 수 있음.

- 계층 관계 표현을 위해 '/'를 사용
- 마지막 문자로 '/' 를 사용하지 않음
- 가독성을 위해 '-'(하이픈)을 사용
- '_' (언더스코어)는 사용하지 않음
- 소문자를 사용
- 파일 확장자를 사용하지 않음

```
1   http://api.example.org/my-folder/my-doc //1
2   HTTP://API.EXAMPLE.ORG/my-folder/my-doc //2
3   http://api.example.org/My-Folder/my-doc //3
```

-> Schem과 HOST에만 대소문자 구별이 없고, 이 외에는 대소문자가 구별.
아래 3개의 URL는 대소문자 구별이 없다면 모두 동일한 URL지만,
1번과 2번은 동일하고 3번은 같지 않음.

3. CRUD 함수명을 사용X

URI는 어떤 동작이 수행되는 지 가르키는 게 아니라, 리소스를 가르키는 것
리소스에 대한 작업은 HTTP Method를 이용하도록 함

```
HTTP GET http://api.example.com/device-management/managed-devices  //Get all devices
HTTP POST http://api.example.com/device-management/managed-devices  //Create new Device

HTTP GET http://api.example.com/device-management/managed-devices/{id}  //Get device
for given Id
HTTP PUT http://api.example.com/device-management/managed-devices/{id}  //Update device
for given Id
HTTP DELETE http://api.example.com/device-management/managed-devices/{id}  //Delete
device for given Id
```

4. 필터를 위해 쿼리 파라미터를 사용

Resource(or Reperesentation)에 대한 정렬, 필터링, 페이징은 신규 API를 생성하지 않고 쿼리 파라미터를 활용.

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices?region=USA
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
http://api.example.com/device-management/managed-devices?
region=USA&brand=XYZ&sort=installation-date
```

참고문서 : REST Resource Naming Guide