REST API 제대로 알고 사용하기

REST API

01

REST의 정의

REST의 정의
REST의 구체적인 개념

REST의 정의

REST는 네트워크 상에서 Client와 Server 사이의 통신 방식 중 하나

"Representational State Transfer"의 약자

자원을 이름(자원의 표현)으로 구분하여 해당 자원의 상태(정보)를 주고 받는 모든 것을 의미

즉, 자원(resource)의 표현(representation) 에 의한 상태 전달

REST의 구체적인 개념

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 의미

CRUD Operation

Create : 생성(POST)

Read : 조회(GET)

Update : 수정(PUT)

Delete : 삭제(DELETE)

HEAD: header 정보 조회(HEAD)

즉, REST는 자원 기반의 구조(ROA, Resource Oriented Architecture) 설계의 중심에 Resource가 있고 HTTP Method를 통해 Resource를 처리하도록 설계된 아키텍쳐를 의미

02

REST 구성

REST 구성

자원(RESOURCE) - URI

행위(Verb) - HTTP METHOD

표현(Representations)

03

REST 의 특징

REST 의 특징

Uniform (유니폼 인터페이스)

URI로 지정한 리소스에 대한 조작을 통일되고 한정적인 인터페이스로 수행하는 아키텍처 스타일

Stateless (무상태성)

무상태성: 작업을 위한 상태정보를 따로 저장하고 관리하지 않음
-> 세션 정보나 쿠키정보를 별도로 저장하고 관리하지 않기 때문에 API 서버는 들어오는 요청만을 단순히 처리
-> 서비스의 자유도가 높아지고 서버에서 불필요한 정보를 관리하지 않음으로써 구현이 단순

Cacheable (캐시 가능)

REST의 가장 큰 특징 중 하나: HTTP라는 기존 웹표준을 그대로 사용
-> 웹에서 사용하는 기존 인프라를 그대로 활용이 가능
-> HTTP가 가진 캐싱 기능이 적용 가능: HTTP 프로토콜 표준에서 사용하는 Last-Modified태그나 E-Tag를 이용하면 캐싱 구현이 가능

REST 의 특징

Self-descriptiveness (자체 표현 구조)

REST의 또 다른 큰 특징 중 하나: REST API 메시지만 보고도 이를 쉽게 이해 할 수 있는 자체 표현 구조로 되어 있다는 것

Client - Server 구조

REST 서버는 API 제공, 클라이언트는 사용자 인증이나 컨텍스트(세션, 로그인 정보)등을 직접 관리하는 구조로 각각의 역할이 확실히 구분됨
-> 클라이언트와 서버에서 개발해야 할 내용이 명확해지고 서로간 의존성이 줄어들게 됨

계층형 구조

REST 서버는 다중 계층으로 구성될 수 있으며 보안, 로드 밸런싱, 암호화 계층을 추가해 구조상의 유연성을 둘 수 있고 PROXY, 게이트웨이 같은 네트워크 기반의 중간매체를 사용할 수 있게 함

04

REST API 디자인 가이드

REST API 디자인 가이드
HTTP METHOD의 알맞은 역할
URI 설계 시 주의할 점
리소스 간의 관계를 표현하는 방법
자원을 표현하는 Colllection과 Document

REST API 디자인 가이드

REST API 설계 시 가장 중요한 항목은 2가지

첫 번째, URI는 정보의 자원을 표현해야 한다.
-> 리소스명은 동사보다는 명사를 사용

잘못된 예시

GET /members/delete/1

URI는 자원을 표현하는데 중점을 두어야 하고, delete와 같은 행위에 대한 표현이 들어가서는 안됨

두 번째, 자원에 대한 행위는 HTTP Method (GET, POST, PUT, DELETE)로 표현

잘못된 예시

GET /members/delete/1

올바른 예시

DELETE /members/1

REST API 디자인 가이드

회원정보를 가져오는 URI

잘못된 예시

GET /members/show/1

올바른 예시

GET /members/1

회원을 추가할 때

잘못된 예시

GET /members/insert/2

- GET 메서드는 리소스 생성에 맞지 않음

올바른 예시

POST /members/2

HTTP METHOD의 알맞은 역할

URI는 자원을 표현하는 데에 집중하고
행위에 대한 정의는 HTTP METHOD를
통해 하는 것
-> REST한 API를 설계하는 중심 규칙

POST, GET, PUT, DELETE 이 4가지의 Method를 가지고 CRUD를 할 수 있음

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

URI 설계 시 주의할 점

1. 슬래시 구분자(/)는 계층 관계를 나타내는 데 사용

http://restapi.example.com/houses/apartments
http://restapi.example.com/animals/mammals/whales

2. URI 마지막 문자로 슬래시(/)를 포함하지 않음

URI에 포함되는 모든 글자는 리소스의 유일한 식별자로 사용되어야 함 -> URI가 다르다는 것은 리소스가 다르다는 것, 역으로 리소스가 다르면 URI도 달라져야 함

REST API는 분명한 URI를 만들어 통신을 해야 하기 때문에
혼동을 주지 않도록 URI 경로의 마지막에는 슬래시(/)를 사용하지 않음

http://restapi.example.com/houses/apartments/ (X)
http://restapi.example.com/houses/apartments (O)

3. 하이픈(-)은 URI 가독성을 높이는데 사용

불가피하게 긴 URI경로를 사용하게 된다면 하이픈을 사용해 가독성을 높일 수 있음

URI 설계 시 주의할 점

4. 밑줄(_)은 URI에 사용하지 않음

밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 함

-> 이런 문제를 피하기 위해 밑줄 대신 하이픈(-)을 사용하는 것이 좋음(가독성 측면)

5. URI 경로에는 소문자가 적합

URI 경로에 대문자 사용은 피하도록 해야 함
-> 대소문자에 따라 다른 리소스로 인식하게 되기 때문

RFC 3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문

RFC 3986 is the URI (Unified Resource Identifier) Syntax document

6. 파일 확장자는 URI에 포함시키지 않음

http://restapi.example.com/members/soccer/345/photo.jpg (X)

REST API에서는 메시지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않음
-> Accept header를 사용

GET / members/soccer/345/photo HTTP/1.1 Host: restapi.example.com Accept: image/jpg

리소스 간의 관계를 표현하는 방법

/리소스명/리소스 ID/관계가 있는 다른 리소스명
ex) GET : /users/{userid}/devices
(일반적으로 소유 'has'의 관계를 표현할 때)

만약에 관계명이 복잡하다면 이를 서브 리소스에 명시적으로 표현하는 방법이 있음

예를 들어 사용자가 '좋아하는' 디바이스 목록을 표현해야 할 경우 다음과 같은 형태로 사용될 수 있음

GET : /users/{userid}/likes/devices
(관계명이 애매하거나 구체적 표현이 필요할 때)

자원을 표현하는 Colllection과 Document

여기서 중요한 점은 컬렉션은 복수로 사용하고 있다는 점
-> 좀 더 직관적인 REST API를 위해서는 컬렉션과 도큐먼트를 사용할 때 단수 복수도 지켜준다면 좀 더 이해하기 쉬운 URI를 설계할 수 있음

DOCUMENT: 문서 또는 한 객체

컬렉션: 문서들의 집합 또는 객체들의 집합

컬렉션과 도큐먼트는 모두 리소스라고 표현할 수 있음
-> URI에 표현됨

sports라는 컬렉션과 soccer라는 도큐먼트로 표현되고 있는 예시

http:// restapi.example.com/sports/soccer

sports, players 컬렉션과 soccer, 13(13번인 선수)를 의미하는 도큐먼트로 URI가 이루어진 예시

http:// restapi.example.com/sports/soccer/players/13

05

HTTP 응답 상태 코드

HTTP 응답 상태 코드

URI는 자원을 표현하는 데에 집중하고
행위에 대한 정의는 HTTP METHOD를 통해 하는 것
-> REST한 API를 설계하는 중심 규칙

잘 설계된 REST API는 URI만 잘 설계된 것이 아닌 그 리소스에 대한 응답을 잘 내어주는 것까지 포함되어야 함

정확한 응답의 상태코드만으로도 많은 정보를 전달할 수가 있기 때문에 응답의 상태코드 값을 명확히 돌려주는 것은 생각보다 중요한 일이 될 수도 있음

상태코드	
200	클라이언트의 요청을 정상적으로 수행함
201	클라이언트가 어떠한 리소스 생성을 요청, 해당 리소스가 성공적으로 생성됨 (POST를 통한 리소스 생성 작업 시)

HTTP 응답 상태 코드

상태코드	
400	클라이언트의 요청이 부적절 할 경우 사용하는 응답 코드
401	클라이언트가 인증되지 않은 상태에서 보호된 리소스를 요청했을 때 사용하는 응답 코드 (로그인 하지 않은 유저가 로그인 했을 때, 요청 가능한 리소스를 요청했을 때)
403	클라이언트가 인증되지 않은 상태에서 보호된 리소스를 요청했을 때 사용하는 응답 코드 (403 보다는 400이나 404를 사용할 것을 권고. 403 자체가 리소스가 존재한다는 뜻이기 때문에)
404	서버는 요청받은 리소스를 찾을 수 없음 브라우저에서는 알려지지 않은 URL을 의미

HTTP 응답 상태 코드

상태코드	
405	클라이언트가 요청한 리소스에서는 사용 불가능한 Method를 이용했을 경우 사용하는 응답 코드
301	클라이언트가 요청한 리소스에 대한 URI가 변경 되었을 때 사용하는 응답 코드 (응답 시 Location header에 변경된 URI를 적어 줘야 함)
500	서버에 문제가 있을 경우 사용하는 응답 코드

Thank you

감사합니다 :)