|  |
| --- |
| **RESTful API** |
| Representational State Transfer |
|  |
|  |
| 인턴 김 지 영 |
|  |
| 2018-09-14 |

|  |
| --- |
|  |

목차

[1. RESTFul API란? 3](#_Toc524947217)

[1.1 REST란? 3](#_Toc524947218)

[1.2. API란? 3](#_Toc524947219)

[1.3. Restful API란? 4](#_Toc524947220)

[2. REST 구조 4](#_Toc524947221)

[3. REST 특징 4](#_Toc524947222)

[4. REST 장점&단점 6](#_Toc524947223)

[4.1. REST 장점 6](#_Toc524947224)

[4.2. REST 단점 7](#_Toc524947225)

[5. REST 사용법 7](#_Toc524947226)

[5.1. REST 기본 사용법(CRUD) 7](#_Toc524947227)

[5.2 REST 안티 패턴 8](#_Toc524947228)

[6. Node.js로 RESTful API 만들기 9](#_Toc524947229)

[7. 참고 10](#_Toc524947230)

# RESTFul API란?

## 1.1 REST란?

REST는 웹의 창시자(HTTP) 중의 한 사람인 Roy Fielding의 2000년 논문에 의해서 소개되었다. 웹의 본래 설계의 우수성을 최대한 활용할 수 있는 네트워크 기반의 아키텍쳐이다.

* 웹에 존재하는 모든 자원(DB 자원, 이미지, 동영상)에 고유한 URI를 부여해 활용하는 것이다.
* URI를 이용해 명시된 자원(resource)에 접근하고, 자원(resource)에 어떠한 조작(CRUD)을 할 지 HTTP메서드로 나타내는 방법이다.
* 다양한 클라이언트의 등장으로 Backend하나로 다양한 Device를 지원하기 위해 REST의 필요성이 증대되었다.

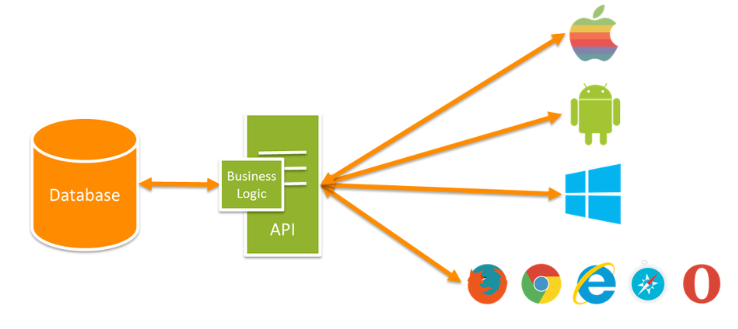
\*URI(Uniform Resource Identifier) : 통합 자원 식별자. 인터넷에 있는 자원을 나타내는 유일한 주소이다.

\*URL(Uniform Resource Locator) : 인터넷 상 자원의 위치를 나타낸다. (=하나의 파일의 위치)

\* URI > URL🡪 URI가 URL의 상위개념이다. Ex) URI(동물)>URL(다람쥐)

## 1.2. API란?

Application Programming Interface. 응용프로그램에서 사용할 수 있도록, 운영체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스를 뜻한다. API를 통해 소스 및 데이터베이스는 접근하지 못하게 하고 해당 프로그램을 사용할 수 있도록 기능을 제공하는 것이다.



[그림1] API 출처: <http://hyunalee.tistory.com/1>

\*Interface : 어떠한 리소스를 토데로 결과물을 만들거나 도출할 떄 쉽게 하기위해 도움을 주는 존재

## 1.3. Restful API란?

**REST(**Representational State Transfer) **특징을** 지키면서 **API**를 제공하는 것을 말한다.

# REST 구조

자원(Resource), 행위(Verb)-HTTP Method, 표현(Representations) 3가지 요소로 구성되어있다.

자원(Resource)-URI

행위(VERB) – HTTP METHOD

* HTTP에는 여러가지 메서드가 있지만 REST에서는 CRUD(Create Read Update Delete)만 사용한다.

|  |  |
| --- | --- |
| 메서드 | 의미 |
| POST | Create |
| GET | Select |
| PUT | Update |
| DELETE | Delete |

[표1] HTTP METHOD

표현(Representations)-Message

* 자원에 대한 행위의 내용을 정의한다. 일반적으로 JSON문서를 이용해 표현되는 실제 데이터이다.

# REST 특징

1. **Uniform(유니폼 인터페이스)**

HTTP 표준에만 따른다면, 안드로이드나 IOS 등 어떤 플랫폼이든, 특정 언어나 기술에 종속되지 않고 모든 플랫폼에 사용이 가능하며, URI로 지정한 리소스에 대한 조작이 가능한 아키텍처 스타일이다.

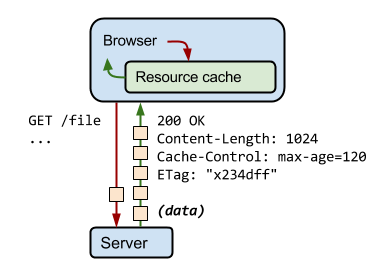
1. **Stateless(무상태성)**

클라이언트의 정보를 서버 쪽에 유지하지 않는다. 세션이나 쿠키 등을 별도로 관리하지 않기 때문에 API서버는 들어오는 요청만 메시지로 처리하기 때문에 구현이 단순하다.

1. **Cacheable(캐시 처리 가능)**

REST는 HTTP라는 기존 웹 표준을 그대로 사용한다. 따라서 HTTP가 가진 캐싱 기능이 적용 가능하다. HTTP 프로토콜 표준에서 사용하는 Last-Modified태그나 E-Tag를 이용하면 캐싱 구현이 가능하다.

\***캐시** : 자주 쓰이는 문서의 사본을 자동으로 보관하는 HTTP장치. 웹 요청이 캐시에 도착했을 때, 캐시된 로컬 사본이 존재한다면, 그 문서는 원 서버가 아니라 그 캐시로부터 제공된다. →웹 콘텐츠를 전송하는 속도를 높여준다.



[그림2] HTTP캐싱

1. **Self-descriptiveness(자체 표현 구조)**

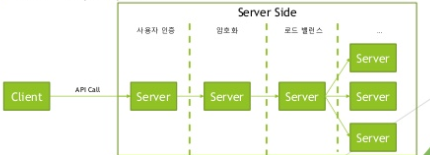
REST API 메시지만 보고도 쉽게 이해 할 수 있는 자체 표현 구조로 되어 있다.

1. **Client – Server Architecture(클라이언트 – 서버 구조)**

REST 서버는 API를 제공하고, 제공된 API를 이용해서 비즈니스 로직 처리 및 저장을 책임진다. 클라이언트의 경우 사용자 인증이나 세션 정보 등을 직접 관리하고 책임진다. 따라서 서로간의 의존성이 줄어들게 된다.

1. **계층형 구조**

REST 서버는 다중 계층으로 구성될 수 있다. 예를 들어, 보안, 로드 밸런싱, 암호화, 사용자 인증 등등 추가하여 구조상의 유연성을 줄 수 있다.



[그림3]REST 계층형 구조 출처 : https://www.slideshare.net/Dreamyn/ss-55676843

# REST 장점&단점

## 4.1. REST 장점

1. 사용이 쉽다.

REST API메시지 자체를 그냥 읽기만 하는 것으로도 메시지의 본래 의도를 파악할 수 있을 정도로 쉽게 이해할 수 있다. 또한, 무상태성이라는 특징덕분에 세션과 요청 내용이 연관되지 않아, 요청이 수행에 독립적이다. 따라서 이해와 사용이 쉽니다.

1. Client-Server가 명확히 분리된다.

클라이언트는 REST API를 통해 서버와 정보를 주고 받는다. 따라서 서버에서는 별도의 클라이언트 정보를 유지할 필요가 없고, 단지 클라이언트에서 요청한 내용만 처리해서 응답하면 된다.

1. 원하는 데이터 표현을 사용할 수 있다.

REST API는 헤더부분에 URI에 대한 처리 메소드를 명시하고, 필요한 실제 데이터는 body부분에 표현하도록 분리시켰다. body부분은 JSON, XML 등 원하는 표현 언어로 사용할 수 있다.

## 4.2. REST 단점

1. HTTP 메소드의 한계에 묶인다.

REST API는 HTTP메소드를 이용하여 URI에 대한 행위를 정의하기때문에 간단한 수준의 메소드만 지원할 수 있다.

1. 표준이 없어서 관리하기 어렵다.

REST API는 API 설계 가이드일 뿐이지 명시적인 표준이 아니기 때문에 관리가 어렵다.

1. RDBMS에 적극적으로 사용할 수 없다.

NoSQL쪽이 더 잘 맞다.

\*NoSQL : 비관계형 데이터베이스. 특정 데이터 모델에 대해 특정 목적에 맞추어 구축되는 데이터베이스로서 현대적인 애플리케이션 구축을 위한 유연한 스키마를 갖추고 있다.

1. 사용할 수 있는 메소드가 4가지 밖에 없다.

CRUD 4가지만을 가지고 모든 행위들을 표현할 수 없다. Control성이나 함수성의 의미를 갖는 것들을 표현해야 하는데 어려움이 있다. 이런 경우에는 PUT이나 POST를 사용하거나 또는 함수에 대한 의미를 재해석하는 접근이 필요하다.

Ex) Send Mail (메일을 보낸다)-> 누구한테 보내는 메일을 생성한다.

# REST 사용법

## REST 기본 사용법(CRUD)

HTTP 메서드를 이용해서 CRUD를 구현하고 메시지를 JSON으로 표현하여 HTTP Body에 실어 보내면 된다.

* 생성 : HTTP Post, <http://myweb/users/>

{ “name”: “terry”,

“address”:”seoul”}

* + - * + Name이 terry 이고 address가 seoul 인 리소스를 생성한다.
        + 다른 메서드들과 달리 URI에 리소스 ID가 없다.
* 조회 : HTTP Get, <http://myweb/users/terry>
  + - * + Id가 terry인 리소스를 조회한다.
* 수정 : HTTP PUT, <http://myweb/users/terry>

{ “name”:”terry”,

“address”:”Suwon”}

* + - * + Id가 terry인 리소스의 내용을 수정한다.
* 삭제 : HTTP DELETE, <http://myweb/users/terry>
  + - * + Id가 terry인 리소스를 삭제한다.

## REST 안티 패턴

### GET/POST를 이용한 터널링

* GET을 이용한 터널링

HTTP GET, <http://myweb/users?method=update&id=terry>

GET을 이용한 터널링이다. 리소스를 업데이트 하는 과정에서, HTTP PUT을 사용하지 않고 GET을 사용한 것이다.

* POST를 이용한 터널링

HTTP POST, <http://myweb/users/>

{ "getuser":{

"id":"terry", }}

생성을 하기 위해서가 아닌 다른 목적으로 POST를 사용하는 경우이다. 위 예시의 경우, id가 terry인 정보를 조회하기 위해선 GET을 이용해야 한다.

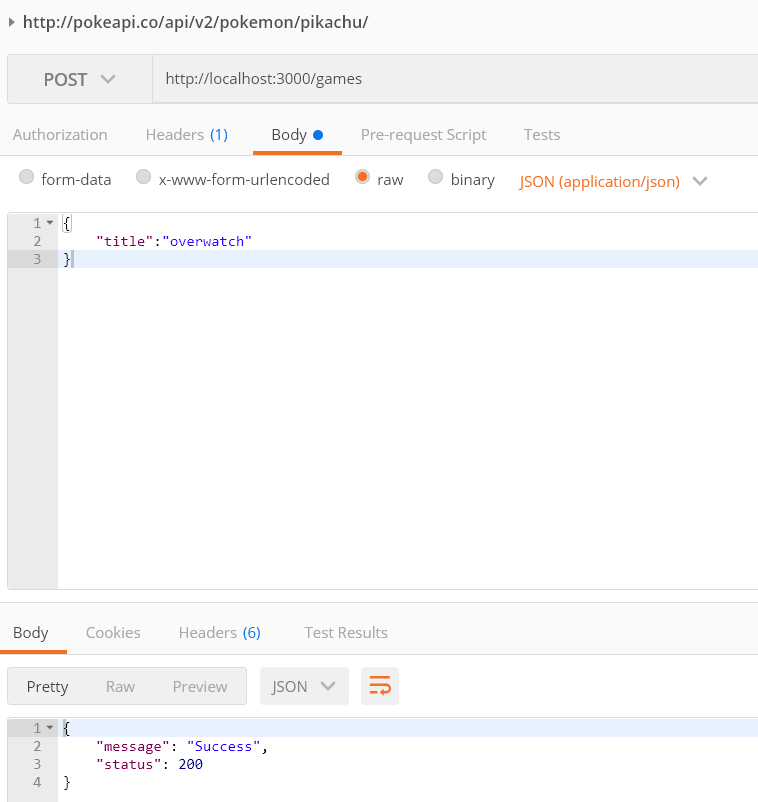
1. HTTP Response code 를 사용하지 않음

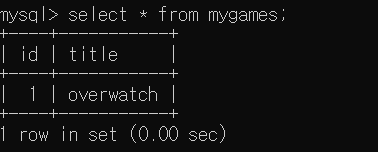
Http Response code를 충실하게 따르지 않고, 성공은 200, 실패는 500 과 같이 1~2개의 HTTP response code만 사용하는 경우이다.

# Node.js로 RESTful API 만들기

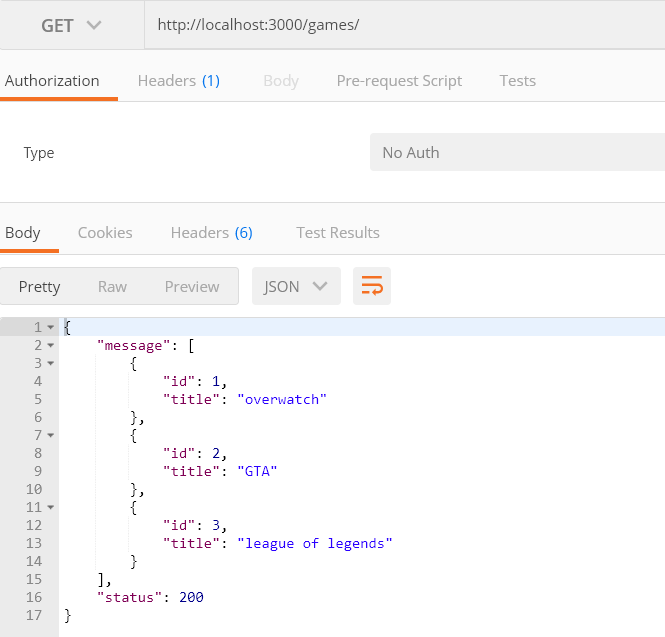
-Postman 이용

POST – 데이터 생성하기(create)

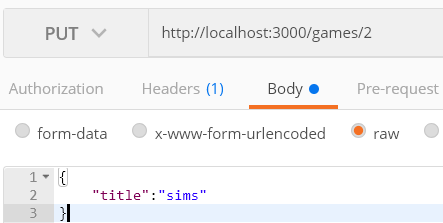




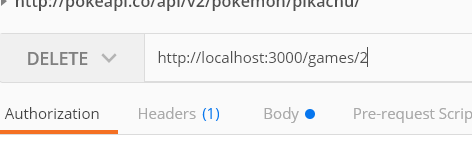
GET-데이터 가져오기 (select)

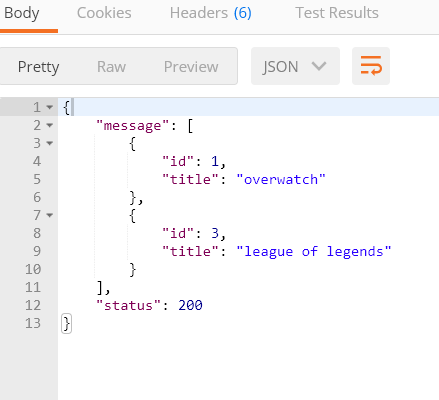


PUT – 데이터 변경하기(Update)



DELETE – 데이터 삭제하기



결과 : 

# 참고

<http://joonyon.tistory.com/13>

<http://bcho.tistory.com/953?category=252770>

<http://jizard.tistory.com/94>

<http://blog.naver.com/PostView.nhn?blogId=complusblog&logNo=220986337770>

<https://ko.wikipedia.org/wiki/%EC%95%84%ED%8B%80%EB%9D%BC%EC%8B%9C%EC%95%88>

<https://aws.amazon.com/ko/what-is-aws/>

<http://bcho.tistory.com/321>

<https://www.slideshare.net/Dreamyn/ss-55676843>