REST

주차	17주차
스터디 일자	@2024/02/07

REST란?

> Representational State Transfer의 약자

자원을 **URI**로 표시하고 해당 자원의 **상태**를 주고 받는 것을 의미

REST의 구성 요소는

- **자원(resource)** : URI
- **행위(Verb)** : HTTP Method
- **표현(Representations)**

으로 이루어짐.

> 💡 즉, REST는 URI를 통해 자원을 표시하고, HTTP Method를 이용해 해당 자원의 행위를 정해주며 그 결과를 받는 것을 말한다!

REST 특징

1. **Uniform Interface (인터페이스 일관성)**

HTTP 표준만 따른다면 어떤 언어 혹은 어떤 플랫폼에서 사용하여도 사용이 가능한 인터페이스 스타일
안드로이드 플랫폼, IOS 플랫폼 등 특정 언어나 플랫폼에 종속되지 않고 사용이 가능

2. **Stateless (무상태)**

   REST는 상태 정보를 유지하지 않음.
   서버는 각각의 요청을 완전히 다른 것으로 인식하고 처리
   이전 요청이 다음 요청 처리에 연관이 되면 안됨.

3. **Cacheable (캐시 처리 가능)**

   HTTP의 기존 웹 표준을 그대로 사용하기 때문에 HTTP가 가진 캐싱 기능 적용 가능

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

4. **Self-descriptiveness (자체 표현 구조)**

   REST API 메시지만 보고도 쉽게 이해할 수 있는 자체 표현 구조로 되어있음.

5. **Client-Server (서버-클라이언트 구조)**

   REST 서버는 API 제공을 하고 클라이언트는 사용자 인증에 관련된 일들을 직접 관리
   자원이 있는 쪽을 Server라고 하고 자원을 요청하는 쪽이 Client가 됨.
   서로간의 의존성이 줄어들기 때문에 역할이 확실하게 구분되어 개발해야할 내용들이 명확해짐.

6. **Layerd System (계층화)**

   클라이언트는 REST API 서버만 호출
   REST 서버는 다중 계층으로 구성될수 있으며, 로드 밸런싱, 암호화, 사용자 인증 등을 추가하여 구조상의 유연성을 둘 수 있음.

7. Code on Demand (optional)

서버가 클라이언트에서 실행시킬 수 있는 로직을 전송하여 클라이언트의 기능을 확장시킬 수 있음.
이를 통해 클라이언트가 사전에 구현해야 하는 기능의 수를 줄여 간소화시킬 수 있음.

but, 현재는 잘 사용되지 않음.
이유 →
https://dr-dev.tistory.com/m/19

REST 장단점

장점

1. HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구축할 필요가 없음.
2. HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해줌.
3. HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능.
4. REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있음.
5. 여러가지 서비스 디자인에서 생길 수 있는 문제를 최소화
6. 서버와 클라이언트의 역할을 명확하게 분리

단점

1. 안티패턴으로 설계될 가능성이 높음.

   REST API의 안티패턴이란 REST API의 특징을 이해하지 못하고 REST 사상에 어긋나는 패턴을 적용한 API들을 말함.
   예) HTTP Method의 잘못된 사용, HTTP Response code를 2, 3개만 활용

2. 표준 규약이 없음.

REST API 설계 규칙

1. URI는 **정보의 자원**을 표현해야 한다.

   자원의 이름은 동사보다는 **명사를 사용**한다.

   URI는 자원을 표현하는데 중점을 두어야 하기 때문에 **행위에 대한 표현이 들어가면 안된다.**
   (URI에 HTTP Method 행위에 대한 동사 표현이 들어가면 안된다.)

   ```
   GET /users/1
   ```

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

   URI에 자원의 행위에 대한 표현이 들어가지 않는 대신 HTTP Method를 통해 대신한다.

   ```
   GET /users/1 ID 1을 가진 유저 정보를 가져오기
   DELETE /users/1 ID 1을 가진 유저 정보를 삭제하기
   POST /users 유저를 생성하기
   ```

3. 슬래시(/)는 **계층 관계**를 나타내는데 사용한다.

   계층의 마지막은 항상 **엘리먼트(요소, 리소스)**가 되어야 한다.

   ```
   http://주소주소주소주소/users/records
   http://주소주소주소주소/users/schedules
   ```

4. URI 마지막은 슬래시(/)를 사용하면 **안된다.**

   ```
   http://주소주소주소주소/users/schedules/ [X]
   http://주소주소주소주소/users/schedules  [O]
   ```

5. 하이픈(-)은 URI의 가독성을 높이는데 사용한다.

불가피하게 긴 URI를 사용하게 될 경우 하이픈을 이용하여 가독성을 높인다.

6. 언더바(_) 혹은 밑줄은 URI에 사용하지 않는다.

   밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 한다.

   그래서 대신 언더바를 사용하지 않고 하이픈을 이용한다.

7. URI는 경로에는 소문자가 적합하다.

   URI 경로에는 대문자 사용을 피해야한다.

   대소문자에 따라 각자 다른 리소스로 인식하기 때문

   RFC3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문

8. 파일 확장자는 URI에 포함하지 않는다.

   REST API에서는 메시지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않는다.

RESTful이란?

REST API의 설계 의도를 명확하게 지켜주는 것이라는 의미

- **REST API**는 REST 기반의 규칙들을 지켜서 설계된 API

RESTful한 API란, *각 구성요소들의 역할이 완벽하게 분리되어 있는 것이라고 할 수 있겠다!*

RESTful API는 누군가가 공식적으로 발표한 것이 아니고 그저 개발자들이 비공식적으로 의견을 제시한 것이라 명확한 정의는 없다.

하지만, RESTful API의 목적은 이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것이다.

CRUD	HTTP METHOD	URI
user들을 표시	GET	/users
user 하나만 표시	GET	/users/:id
user를 생성	POST	/users
user를 수정	PUT	/users:id
user를 삭제	DELETE	/users:id

RESTful 하지 못한 경우?

- CRUD 기능을 전부 POST Method로만 처리하는 API
- URI에 자원과 id 외 정보가 들어가는 경우 등

```
PUT /users/update-nickname [X]
PUT /users/:id/nickname [O]
```