REST, RESTful의 개념

{ REST : API }

Representational State Transfer API

항해99 14기 이준영

1. REST의 등장 배경

REST의 등장 배경

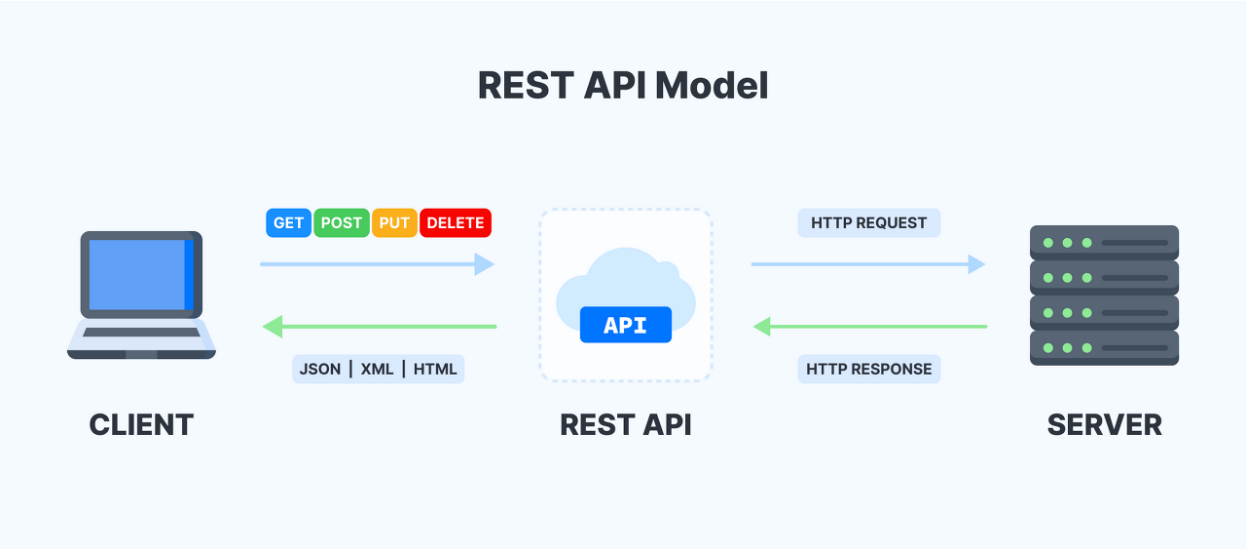
1. REST의 등장 배경

1. REST의 등장

REST는 **인터넷과 같이 복잡한 네트워크 통신이 등장함에 따라, 이를 관리하기 위한 지침으로 만들어졌다.** 대부분의 비즈니스 애플리케이션은 다양한 태스크를 수행하기 위해 다른 내부 애플리케이션 및 서드 파티 애플리케이션과 통신해야 한다.

> **어플리케이션의 통합, 멀티 플랫폼**
> 최근의 서버 프로그램은 다양한 브라우저와 안드로이드폰, 아이폰과 같은 모바일 디바이스에서도 통신을 할 수 있어야한다. 이러한 **멀티 플랫폼에 대한 지원을 위해 서비스 자원에 대한 아키텍처를 세우고 이용하는 방법을 모색**한 결과, REST에 관심을 가지게 되었다.

REST의 등장 배경



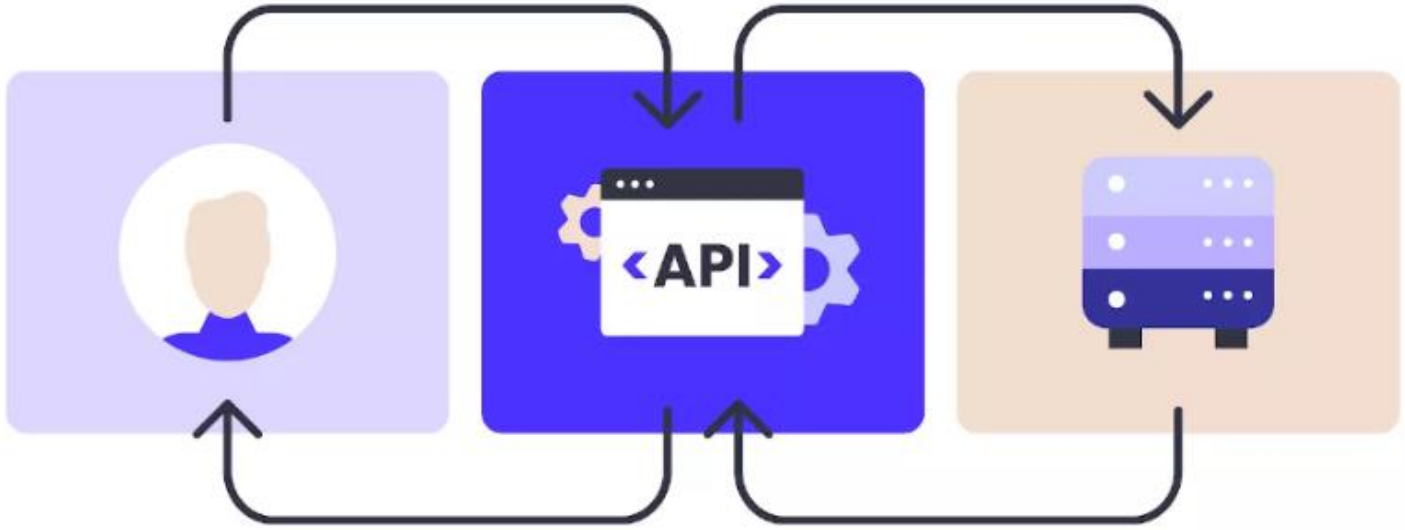
이 때 REST 기반 아키텍처를 사용하여 **대규모의 고성능 통신을 안정적으로 지원**할 수 있다. 쉽게 구현하고 수정할 수 있어 모든 API 시스템을 파악하고 여러 플랫폼에서 사용할 수 있다.

REST API : 클라이언트와 서버 간의 두 컴퓨터 시스템이 인터넷을 통해 정보를 안전하게 교환하기 위해 사용하는 인터페이스

API란?

2. API란 무엇?

웹 API는 **클라이언트와 웹 리소스 사이의 네트워크 통신을 위한 게이트웨이**라고 생각할 수 있다. 애플리케이션 프로그래밍 인터페이스(API)는 다른 소프트웨어 시스템과 통신하기 위해 따라야 하는 규칙을 정의한다.



2. REST란 무엇인가?

REST란 무엇인가?

2. REST란 무엇인가?

REST(Representational State Transfer)의 약자

: 1. 자원을 이름으로 구분하여 2. 해당 자원의 상태를 주고받는 모든 것을 의미

{ REST : API }

Representational State Transfer API

REST란 무엇인가?

> 'HTTP URI(Uniform Resource Identifier)'를 통해 **자원(Resource)을 명시**하고,
>
> 'HTTP Method(POST, GET, PUT, DELETE, PATCH 등)'를 통해 해당 자원(URI)에 대한 **'CRUD Operation'을 적용**

💡 CRUD Operation이란?

Create : 데이터 생성(POST)
Read : 데이터 조회(GET)
Update : 데이터 수정(PUT, PATCH)
Delete : 데이터 삭제(DELETE)

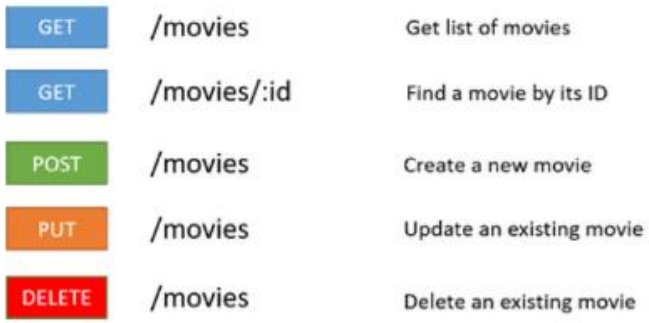
REST란 무엇인가?

1. 자원을 이름으로 구분? = 자원의 표현

자원 = 문서, 사진, 그림, 데이터 등 소프트웨어가 관리하는 모든 것을 HTTP URI(Uniform Resource Identifier)를 통해 명시

Ex) DB의 영화 정보가 자원일 때, /movies를 자원의 표현으로 정한다.

```
GET /movies/1
```



REST란 무엇인가?

2. 자원의 상태를 주고 받음(요청 -> 응답)

클라이언트는 데이터가 요청되어지는 시점에서 자원의 상태(정보= payload)를 전달한다.

Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보낸다.

=> **전달 방식으로는 JSON 혹은 XML**를 통해 데이터를 주고 받는 것이 일반적

즉, REST는 HTTP URI(Uniform Resource Identifier)를 통해 Client와 Server 사이의 통신하는 방식 중 하나로,

HTTP Method(POST, GET, PUT, DELETE, PATCH 등)를 통해 자원을 처리(CRUD)하도록 설계된 아키텍쳐를 말한다.

REST의 구성요소

https:/www.naver.com/books/10

/books

GET	/books	Lists all the books in the database
DELETE	/books/{bookId}	Deletes a book based on their id
POST	/books	Creates a Book
PUT	/books/{bookId}	Method to update a book
GET	/books/{bookId}	Retrieves a book based on their id

{JSON}

1. 자원(Resource) : HTTP URI

- 모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재한다.
- 자원을 구별하는 ID는 '/groups/:group_id'와 같은 HTTP URI

Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)에 대한 조작을 Server에 요청한다.

2. 행위(Verb): HTTP Method

HTTP 프로토콜의 Method를 사용한다.

HTTP 프로토콜은 GET, POST, PUT, DELETE 와 같은 메서드를 제공한다.

3. 표현(Representation of Resource)

Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보낸다.

REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 응답을 받을 수 있다.

JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적이다.

REST의 특징과 장단점

REST의 특징

1) Uniform (유니폼 인터페이스)

Uniform Interface는 URI로 지정한 리소스에 대한 조작을 통일되고 한정적인 인터페이스로 수행하는 아키텍처 스타일을 말합니다.

2) Stateless (무상태성)

REST는 무상태성 성격을 갖습니다. 다시 말해 작업을 위한 상태정보를 따로 저장하고 관리하지 않습니다. 세션 정보나 쿠키정보를 별도로 저장하고 관리하지 않기 때문에 API 서버는 들어오는 요청만을 단순히 처리하면 됩니다. 때문에 서비스의 자유도가 높아지고 서버에서 불필요한 정보를 관리하지 않음으로써 구현이 단순해집니다.

3) Cacheable (캐시 가능)

REST의 가장 큰 특징 중 하나는 HTTP라는 기존 웹표준을 그대로 사용하기 때문에, 웹에서 사용하는 기존 인프라를 그대로 활용이 가능합니다. 따라서 HTTP가 가진 캐싱 기능이 적용 가능합니다. HTTP 프로토콜 표준에서 사용하는 Last-Modified태그나 E-Tag를 이용하면 캐싱 구현이 가능합니다.

4) Self-descriptiveness (자체 표현 구조)

REST의 또 다른 큰 특징 중 하나는 REST API 메시지만 보고도 이를 쉽게 이해 할 수 있는 자체 표현 구조로 되어 있다는 것입니다.

5) Client - Server 구조

REST 서버는 API 제공, 클라이언트는 사용자 인증이나 컨텍스트(세션, 로그인 정보)등을 직접 관리하는 구조로 각각의 역할이 확실히 구분되기 때문에 클라이언트와 서버에서 개발해야 할 내용이 명확해지고 서로간 의존성이 줄어들게 됩니다.

6) 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 합니다.

REST의 특징과 장단점

장점

- HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구출할 필요가 없다.

- HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해준다.

- HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.

- Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.

- REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있다.

- 여러가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.

- 서버와 클라이언트의 역할을 명확하게 분리한다.

단점

- 표준이 존재하지 않는다.

- 사용할 수 있는 메소드가 4가지 밖에 없다.HTTP Method 형태가 제한적이다.

- 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 값이 왠지 더 어렵게 느껴진다.

- 구형 브라우저가 아직 제대로 지원해주지 못하는 부분이 존재한다..

ex) PUT, DELETE를 사용하지 못하는 점

ex) pushState를 지원하지 않는 점

3. REST API란?

REST API

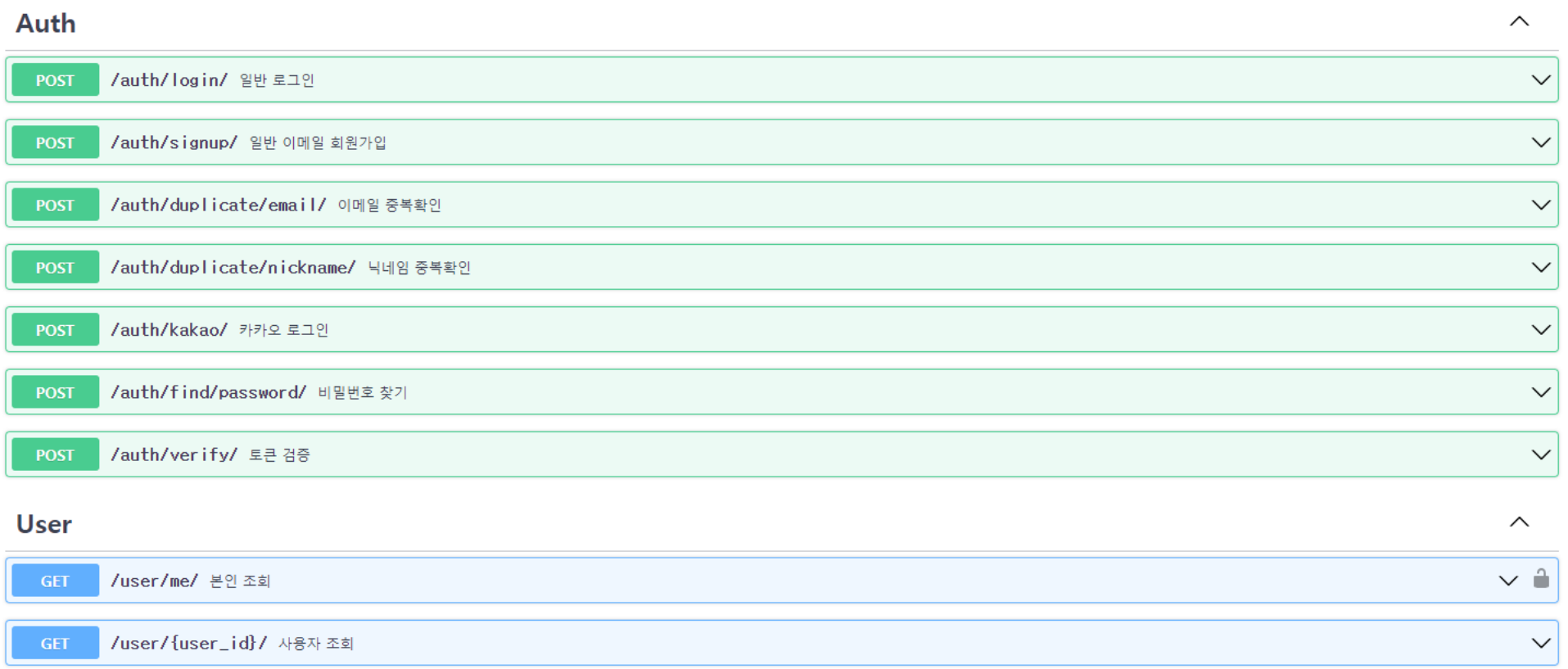
3. REST API

REST API : REST 기반으로 서비스 API를 구현한 것

최근 OpenAPI(누구나 사용할 수 있도록 공개된 API: 구글 맵, 공공 데이터 등), 마이크로 서비스(하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처) 등을 제공하는 업체 대부분은 REST API를 제공한다.

REST API

API와 함께 항상 API명세도 제공되어야 한다. 클라이언트는 URI를 모르면 사용할 수 없다.



REST API 설계 규칙

1. 슬래시 구분자(/)는 계층 관계를 나타내는데 사용

```
http://restapi.example.com/houses/apartments
```

2. URI 마지막 문자로 슬래시(/)를 포함하지 않는다.

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

```
http://example.com/posts/    (X)
```

REST API 설계 규칙

3. 하이픈(-)은 URI 가독성을 높이는데 사용

- 불가피하게 긴 URI경로를 사용하게 된다면 하이픈을 사용해 가독성을 높인다.

4. 밑줄(_)은 URI에 사용하지 않는다.

- 밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 하므로 가독성을 위해 밑줄은 사용하지 않는다.

5. URI 경로에는 소문자가 적합하다. 대문자 사용은 피하도록 한다.

- RFC 3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문
파일확장자는 URI에 포함하지 않는다.

6. REST API에서는 메시지 body 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않는다.
Accept header를 사용한다.

```
http://restapi.example.com/members/345/photo.jpg (X)
GET / members/345/photo HTTP/1.1 Host: restapi.example.com Accept: image/jpg (O)
```

REST API 설계 규칙

7. 리소스 간에는 연관 관계가 있는 경우

=> /리소스명/리소스 ID/관계가 있는 다른 리소스명

```
GET : /users/{userid}/devices (일반적으로 소유 'has'의 관계를 표현할 때)
```

REST API 설계 규칙

요약

1) 명사, 소문자 => 동사x

2) 명사는 복수형

3) URI 마지막은 / 포함x

4) URI는 언더바x 하이픈 사용 -

5) 파일의 확장자를 표시x

3. RESTful 하다?

RESTful

4. RESTful

1. RESTful 하다?

REST 원리를 따르는 시스템은 RESTful이란 용어로 지칭

=> **'REST API'를 제공하는 웹 서비스를 'RESTful'하다고 할 수 있다.**

2. RESTful의 목적

=> **이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것**

RESTful한 API를 구현하는 근본적인 목적이 성능 향상에 있는 것이 아니라 **'일관적인 컨벤션을 통한 API의 이해도 및 호환성을 높이는 것'**이 주 동기이니, 성능이 중요한 상황에서는 굳이 RESTful한 API를 구현할 필요는 없다.

RESTful하지 못한 경우

3. RESTful 하지 못하는 경우

URI 규칙을 올바르게 지키지 않은 API는 REST API의 설계 규칙을 올바르게 지키지 못한 시스템은 REST API를 사용하였지만 RESTful 하지 못한 시스템이라고 할 수 있다.

> Ex1) CRUD 기능을 모두 POST로만 처리하는 API
> Ex2) route에 resource, id 외의 정보, URI에 행위(method)에 대한 부분이 들어가는 경우 (/students/updateName)

Path Variable vs Query Parameter

5. Client에서 Rest API에 요청을 보내는 방식 : Path Variable vs Query Parameter

Path Variable

```
GET /users/10
```

전체 데이터 또는 특정 하나의 데이터를 다룰 때 처럼, 리소스를 식별하기 위해 사용

Query Parameter

```
GET /users?user_id=10
```

데이터의 좀 더 깊은 속성 값을 조정하거나, 세밀하게 데이터를 정렬하거나 필터링 하고 싶을 경우 더 적합하다.

따라서 API를 개발할 때에는 PATH방법과 QUERY방법을 필요에 따라 적절하게 사용하도록 하며, 데이터 요청 시 API 명세에 따르도록 한다. API 명세는 협업 시에 프론트와 백엔드가 잘 협업하여 설계하도록 한다.

감사합니다!