[Network] HTTP, HTTPS, REST API

HTTP 개념

- HyperText Transfer Protocol의 약자
- 웹 상에서 서버와 클라이언트 간에 request/response 구조로 정보를 주고받을 수 있는 프로토콜

HTTP 특징

- Connectionless (비연결성)
  - 클라이언트가 서버로 요청을 보낸 후에 서버로부터 응답을 받으면 연결을 끊어버린다.
  - 이로 인해 많은 사람이 웹을 이용하더라도 실제 동시 접속을 최소화하여 더 많은 유저의 요청을 처리할 수 있습니다.
- Stateless (무상태성)
  - Connectionless한 속성을 가지지만, 연결을 끊었기 때문에 클라이언트의 이전 상태(로그인 유무 등)를 알 수가 없다

HTTPS

- HTTP에 암호화를 추가한 프로토콜 (보안 강화!)
    - HTTP는 정보를 텍스트 형식으로 주고받기 때문에 중간에 인터셉트할 경우 내용이 노출될 수 있는 보안 이슈가 있다. 이를 해결하기 위해 탄생하게 되었다.
- S는 Over Secure Socket Layer의 약자
- HTTP 통신이 SSL(TLS) Layer 위에서 이루어 지는 것, 쉽게 말하면 어떤 보안 체계 위에서 이루어지는 HTTP를 HTTPS라 한다.
    - SSL
        - Secure Sockets Layer의 약자
        - 인터넷에서 데이터를 안전하게 전송하기 위한 통신 규약 프로토콜이다.
            - SSL 위에서 HTTP가 동작하면 HTTPS가 되고, FTP가 동작하면 SFTP가 된다.

HTTP Request Method

- GET
    - 서버에 존재하는 데이터를 요청 (조회)
- POST
    - 서버에 데이터를 생성하는 것을 요청 (생성)
- PUT
    - 서버에 존재하는 데이터를 수정하거나 존재하지 않으면 생성 (생성, 수정)
- DELETE
    - 서버에 데이터를 제거할 것을 요청 (삭제)
        - 존재하지 않아도 동일하게 동작한다.
- PATCH
    - 서버에 존재하는 데이터를 일부 수정
    - PUT과 유사하나, 모든 데이터를 갱신하는 것이 아닌 리소스의 일부분만 수정할 때 쓰인다.

- GET vs POST 비교

특징	GET 방식	POST 방식
캐시화(cached)	캐시될 수 있음.	캐시되지 않음.
브라우저 히스토리	히스토리에 쿼리 문자열이 기록됨.	히스토리에 기록되지 않음.
데이터 길이	데이터의 길이가 URL 주소의 길이 이내로 제한됨. (익스플로러에서 URL 주소가 가질 수 있는 최대 길이는 2,083자이며, 이 중에서 순수 경로 길이는 2,048자까지만 허용됨)	제한 없음.
데이터 타입	오직 ASCII 문자 타입의 데이터만 전송할 수 있음.	제한 없음.
보안성	데이터가 URL 주소에 포함되어 전송되므로, 아무나 볼 수 있어 보안에 매우 취약함.	브라우저 히스토리에도 기록되지 않고, 데이터가 따로 전송되므로, GET 방식보다 보안성이 높음.

  - GET
    - GET은 가져오는 것! 서버의 값이나 상태 등을 변경하지 않는다.
    - 요청하는 데이터가 HTTP Request Message의 Header 부분에 url이 담겨서 전송된다. 때문에 url 상에 ? 뒤에 데이터가 붙어 request를 보내게 되는 것이다.
    - url 주소에 데이터를 담기 때문에, 전송할 수 있는 데이터의 크기가 제한적이다.
    - 보안이 필요한 데이터에 대해서는 적절하지 않다 (ex. password)
      - 데이터가 그대로 url에 노출되고,
      - 요청이 브라우저에 의해 캐시되어 저장되기 때문
  - POST
    - 서버의 값이나 상태를 변경하기 위해서 또는 추가하기 위해서 사용
    - request를 보낼 때 HTTP Request Message의 Body 부분에 데이터를 담아서 전송한다
    - 브라우저에 의해 캐시되지않으므로 브라우저 히스토리에도 남지않는다
    - 따라서 데이터 크기가 GET 방식보다 크고 보안성이 (상대적으로) 높다

HTTP 상태 코드

- **1xx (정보전송 임시 응답)** : 전송 프로토콜 수준의 정보 교환 (요청 후 작업 계속 진행)
- **2xx (성공)** : 클라이언트 요청이 성공적으로 수행됨

- **3xx (리다이렉트)** : 클라이언트는 요청을 완료하기 위해 추가적인 행동을 취해야 함(변경된 url)
- **4xx (클라이언트 에러)** : 클라이언트의 잘못된 요청
- **5xx (서버 에러)** : 서버쪽 오류로 인한 상태코드

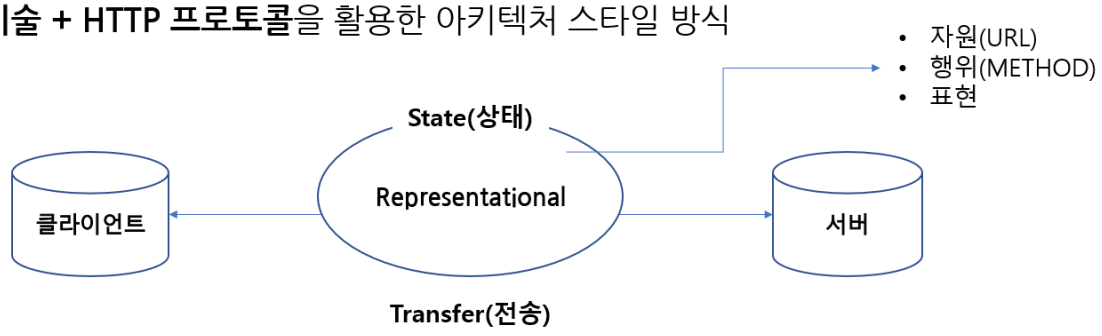
[참고] 자주 등장하는 HTTP 응답코드

status code	message	
200	OK	요청이 성공함 (ex. 잔액조회 성공)
201	Created	리소스 생성 성공 (ex. 게시글 작성 성공, 회원가입 성공)
400	Bad Request	데이터의 형식이 올바르지 않는 등 서버가 요청을 이해할 수 없음 (ex. 올바르지 않은 형식의 데이터 입력 등)
401	Unauthorized	인증되지 않은 상태에서 인증이 필요한 리소스에 접근함 (ex. 로그인 전에 사용자 정보 요청 등)
403	Forbidden	인증된 상태에서 권한이 없는 리소스에 접근함 (ex. 일반 유저가 관리자 메뉴 접근 등)
404	Not Found	요청한 route가 없음. 찾는 리소스가 없음 (ex. www.naver.com/nossi 등 존재하지 않는 route에 요청 등)
502	Bad Gateway	서버에서 예상하지 못한 에러가 발생함 (ex. 예외처리를 하지 않은 오류가 발생 등)

REST API 개념

- 자원을 이름으로 구분해 해당 자원의 상태를 전송하여 주고 받는 것
- **RE**presentational **S**tate **T**ransfer!

WEB 기술 + HTTP 프로토콜을 활용한 아키텍처 스타일 방식



REST API 특징

- **Server-Client**
  - Server : 자원 존재 / Client : 자원 요청
  - 클라이언트와 서버의 의존성 감소
- **Stateless**
  - Client의 context를 Server에 저장하지 않음 → 구현 단순
  - Server는 모든 요청을 각자 별개의 것으로 인식하고 처리
- **Cacheable**
  - HTTP의 캐싱 기능 활용 가능 → 응답 시간, 성능 빨라짐
- **Uniform Interface**
  - Loosely Coupling: 특정 언어에 종속되지 않고 HTTP 프로토콜을 따르면 모든 플랫폼에 적용 가능

REST API 설계

- **Resource**
  - 동사보다 명사, 대문자보다 소문자
  - 객체 인스턴스, DB : 단수 명사
  - 서버 및 클라이언트 리소스 : 복수 명사

REST API를 사용하는 이유 (장점)

- 다양한 클라이언트를 대응할 수 있다!
  - 이젠 웹 서버에서 웹 브라우저, 안드로이드, IOS 등 다양한 멀티 플랫폼에서 통신을 지원할 수 있어야 한다. 이에 필요한 아키텍처로 REST가 통용되기 시작
- HTTP 프로토콜 인프라를 사용하여 추가 인프라 구축이 필요 없다
- API 메시지 의미가 명확하므로 이해하기 쉽다

REST API 단점

- 명확한 표준이 존재하지 않다!
- RESTful을 완전히 만족하는 API를 만들기가 매우 까다롭다.