HTTP & HTTPS

HTTP (80)

- HyperText Transfer Protocol의 약자
- 클라이언트와 서버 사이에 이루어지는 요청 / 응답 프로토콜
- 웹에서 브라우저 ↔ 서버 간에 데이터릴 주고 받기 위한 방식

특징

- Request & Response
  - 클라이언트가 HTTP 요청(Request)을 보내면 요청에 대한 결과를 만들어서 응답(Response)
- Stateless
  - HTTP는 상태가 없는 프로토콜
  - Socket이나 TCP 처럼 계속 연결되어 있는 상태가 아니기 때문에 각각의 통신이 독립적
  - 서버가 클라이언트의 상태를 유지하지 않기 때문에 요청을 보낼 때마다 모든 데이터를 매번 보내야 함
  - 클라이언트에서 서버에 데이터 요청 시 아무 서버나 호출해도 됨
  - 로그인과 같이 상태를 유지해야만 하는 경우 쿠키와 서버 세션 등을 사용해서 상태를 유지

HTTP 메시지 구조

구조	의미
start-line	시작 라인
hedaer	HTTP 헤더
CRLF	공백 라인
message body	HTTP 메세지 바디

- start-line

    - Method : 서버가 수행해야 할 동작 (GET, POST)
    - request-targer : 요청이 전송되는 목표 주소
    - HTTP-version HTTP 버전 명시
    - status-code HTTP 응답에 대한 상대 코드 표현

HTTP Method

- GET : 리소스를 조회
    - URL 입력, 링크 클릭
    - 서버에 데이터 전달하는 경우 Path Variable, Query String 사용
    - 멱등성 보장
- POST : 데이터 추가, 등록
    - 새로운 리소스를 생성 (Create)하는데 사용
    - 데이터 전달 시 Body에 쿼리 파라미터 (Key - Value) 형식으로 전달
    - 데이터가 외부로 노출되지 않아 보안상 이점
    - Body 데이터를 bytes로 변환하는데 있어 시간상 손실 발생
- PUT : 리소스 대체, 수정 / 리소스가 없는 경우 새롭게 생성
    - 리소스를 완전히 대체하는 개념(덮어쓰기)
    - 클라이언트가 리소스를 식별할 수 있음 ex) PUT /posts/1 → 1번 게시글 수정
    - 부분 수정이 불가능
    - 멱등성 보장
- DELETE : 리소스 삭제
    - 리소스 제거
    - 멱등성 지님
- PATCH : 리소스 부분 변경
    - 리소스를 수정하는 역할을 하지만 리소스를 부분 변경한다는 점에서 차이 발생
    - A, B 데이터가 있는 경우 B=C로 대체 후 PATCH 요청 시 데이터가 A, C로 발생

    - 멱등성이 왜 보장되지 않을까?
        - 기존 리소스에 응답을 추가하는 경우에도 PATCH가 사용되며 이런 경우엔 결과 달라짐
- HEAD : GET과 동일하나 HTTP Body 부분 제외하고 조회
- OPTIONS : OPTIONS : 서버와 브라우저가 통신하기 위한 통신 옵션 확인
- CONNECT : 대상 자원으로 식별되는 서버에 대한 연결 요청

멱등성

- 동일한 요청을 반복할 때 처리 결과가 늘 같은 것

Status Code

- 1XX : Informational (정보 제공)
    - 임시 응답으로 현재 요청까지는 처리
- 2XX : Success (성공)
    - 클라이언트의 요청이 서버에서 성공적으로 처리
        - 200 : OK, 서버가 요청 성공적으로 처리
        - 201 : Created, 요청이 처리되어 새로운 리소스 생성
        - 202 : Accepted, 요청 접수, 처리 완료 X
- 3XX : Redirection (리다이렉션)
    - 완전한 처리를 위해 추가 동작이 필요한 경우
    - 서버 주소 또는 요청한 URI 웹 문서 이동으로 다시 시도
- 4XX : Client Error (클라이언트 에러)
    - 없는 페이지를 요청하는 등 클라이언트의 요청 메세지 내용이 잘못된 경우
        - 400 : Bad Request, 요청 구문이 잘못됨
        - 401 : Unauthorized, 지정한 리소스에 대한 액세스 권한 없음
        - 403 : Forbidden, 401 인증 이외의 사유로 지정한 리소스에 대한 액세스가 금지됨

        - 404 : Not Found, 지정한 리소스를 찾을 수 없다
- 5XX : Server Error (서버 에러)
    - 서버 사정으로 메세지 처리에 문제가 발생한 경우
    - 서버 부하, DB 처리 과정 오류
    - 서버 Exception 발생 등
        - 500 : Internal Server Error, 서버 에러 발생
        - 501 : Not Implemented, URL 메소드에 대해 서버가 구현하고 있지 않음
        - 502 : Bad Gateway, 게이트웨이 또는 프록시 역할을 하는 서버가 그 뒷단의 서버로부터 잘못된 응답을 받았다.

HTTP 헤더

- 기본적으로 key : value 목록으로 구성
- HTTP 전송에 필요한 모든 부가 정보
    - content-type
    - cookie
    - JWT 등

HTTP 메세지 바디

- HTTP가 실제 전송할 데이터를 담고 있는 부분
- byte로 표현할 수 있는 모든 데이터 전송 가능
    - JSON
    - xml
    - text
    - form-data 등등

문제점

- 서버에서 브라우저로 전송되는 정보가 암호화되지 않는다.
- 즉 데이터가 쉽게 도난 당할 수 있음

HTTPS (443)

- SSL / TLS를 사용하여 정보를 암호화
- 서버 브라우저가 민감한 정보 ex)암호, 인증키 등을 주고받을 때 이것이 도난당하는 것을 막아줌
- 누군가 중간에 데이터를 훔쳐내도 이를 복호화 하는 키는 서버만이 알고 있기 때문에 해독 불가

SSL (Secure Sockt Layer) / TLS

- 웹 서버와 웹 브라우저간의 암호화 통신을 위해 응용계층과 TCP/IP 계층에서 동작하는 프로토콜
- 인증, 암호화, 무결성, 지원 프로토콜 기능 가짐
- 대칭키 암호화 방식과 비대칭키 암호화 방식을 함께 사용함
    - 비대칭키 암호화 방식의 경우 복잡한 수학적 원리로 이루어져 있어 CPU 리소스를 크게 소모

리버스 프록시

- 서버 실행 시 80이나 443으로 들어오는 요청들을 Nginx, Apach HTTP Server 등을 활용해서
  내부 다른 포트로 전달해줌

장점

- 포트 매핑 및 로드 밸런싱
    - 서버가 80포트를 사용하지 않아도 외부에서 80번 포트로 들어오는 요청을 처리 가능
    - 여러 서버의 트래픽을 분산해서 로드 밸런싱 가능
    - ex) 80번 포트에 서버 1개 사용
    - ex) 80번 포트에 리버스 프록시 활용 및 3000, 3001, 3002 포트에서 총 3개의 서버 운용
- 보안 강화
    - HTTPS 트래픽 처리 및 내부 서버는 HTTP 사용 가능

    - 방화벽 역할을 해서 실제 내부 서버의 포트 번호를 숨길 수 있음
- 접근 제어 및 로깅 가능