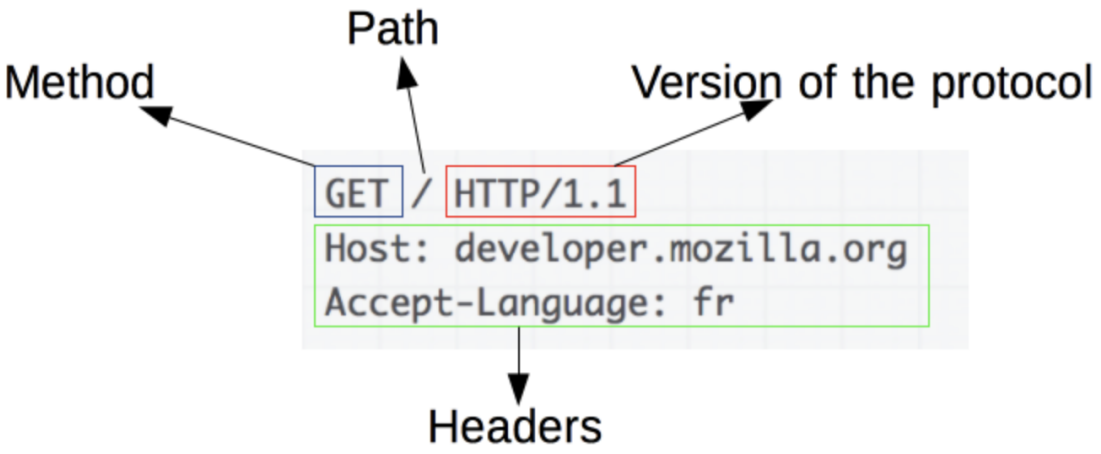
HTTP / HTTPS

HTTP

- **정의**

HTTP(Hyper Text Transfer Protocol)란 서버/클라이언트 모델을 따라 데이터를 주고 받기 위한 **프로토콜**이다. 즉, HTTP는 인터넷에서 하이퍼텍스트를 교환하기 위한 통신 규약으로, 80번 포트를 사용하고 있다. 따라서 HTTP 서버가 80번 포트에서 요청을 기다리고 있으며, 클라이언트는 80번 포트로 요청을 보내게 된다.

- **구조**



- **특징**
  - **HTTP 메시지**에 모든것을 전송한다.
  - HTML,TEXT,이미지,파일,API... 서버간에 데이터를 주고 받을 때도 HTTP 사용
  - 가장 많이 사용하는 버전은 HTTP/1.1 , 현재 2,3도 나왔으며 2까지는 TCP, 3부터는 UDP를 사용한다.
  - 클라이언트 서버 구조 / **무상태 프로토콜(stateless)** / **비연결성 /** HTTP 메시지 사용 / 단순함, 확장 가능

무상태 프로토콜

서버가 클라이언트의 상태를 보존하지않음

서버의 확정성이 높으나 클라이언트가 추가 데이터를 전송해야함

- Stateful과 Stateless의 차이
  - Stateful

  서버가 클라이언트의 상태를 계속 알고 있어야 한다.

  > 이 노트북 얼마에요? / 2개 구매 할게요 / 신용카드로 구매 할게요
  >
  > 같은 점원(서버)이 계속 대화를 하면 위의 대화가 성립한다.(점원이 클라이언트가 이전에 무엇을 물어봤는지 알고있어 대화가 성립.)
  >
  > 만약 각 질문마다 다른 점원이 대답을 한다 가정하자. 두번째 질문을 받은 점원은 무엇을 2개 산다는지 알 수 없다. 세번째 점원도 무엇을 몇 개 신용카드로 구매할지 알 수 없다.
  >
  > **즉 중간에 서버가 문제가 생겨 다른 서버로 바뀌면 대화를 처음부터 다시 진행해야한다.**

  - Stateless

  서버가 클라이언트의 상태를 모르고 있어도 된다.

  > 이 노트북 얼마에요? / 노트북 2개 구매할게요 / 노트북 두개 신용카드로 구매할게요
  >
  > 각 질문을 다른 점원이 받아도 문제 없이 응답할 수 있다.
  >
  > **즉 중간에 서버가 바뀌어도 계속 진행할 수 있다**

- Stateless의 장점
  - 갑자기 고객이 증가해도 점원을 대거 투입할 수 있다.
  - 갑자기 클라이언트 요청이 증가해도 서버를 대거 투입할 수 있다.
  - 무상태는 응답 서버를 쉽게 바꿀 수 있다. -> **무한한 서버 증설 가능**
- Stateless의 한계
  - 모든 것을 무상태로 설계 할 수 있는 없는 경우도 있다.
  - 무상태 예시) 로그인이 필요 없는 단순한 서비스 소개 화면

- 상태 유지예시) 로그인한 사용자의 경우 로그인 했다는 상태를 서버에 유지
- 일반적으로 브라우저 **쿠키와 서버 세션**등을 사용해서 상태 유지
- 상태 유지는 최소한만 사용

비 연결성(connectionless)

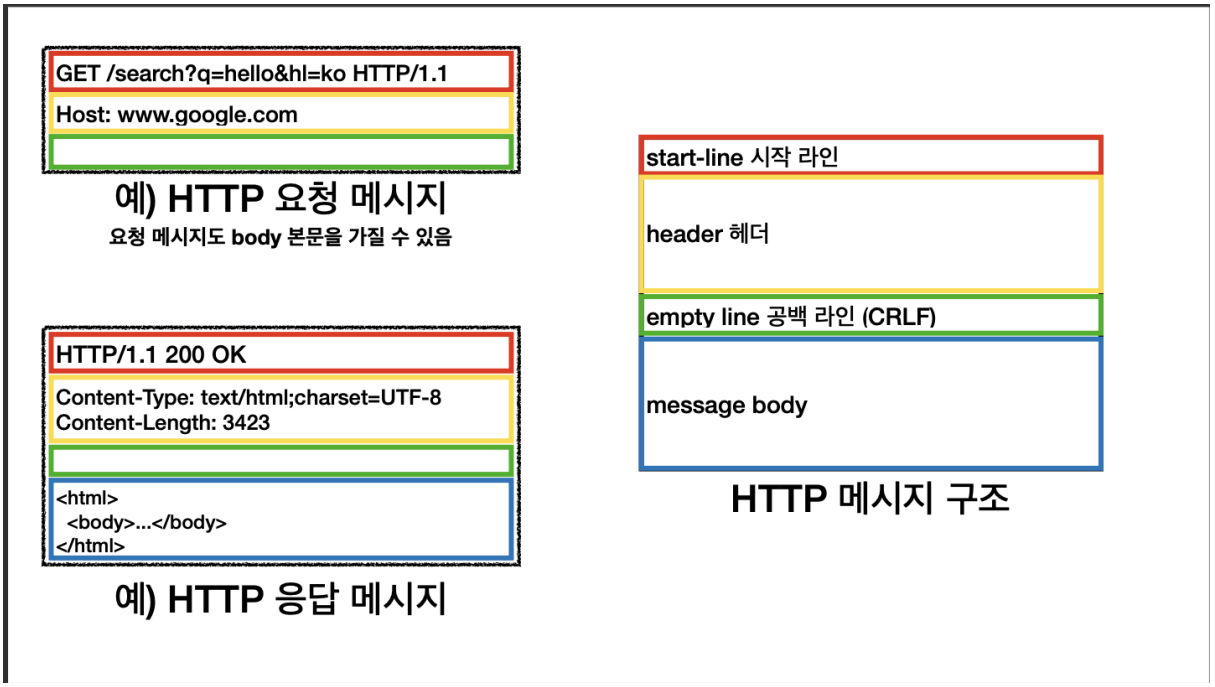
- HTTP는 기본이 연결을 유지하지 않는 모델이다.
- 일반적으로 초 단위 이하의 빠른 속도로 응답
- 1시간 동안 수천명이 서비스를 사용해도 실제 서버에서 동시에 처리하는 요청은 수십개 이하로 매우 적다. 이는 모든 사용자가 동시에 검색버튼을 누르거나 하지 않기 때문이다.
- 서버 자원을 효율적으로 사용 가능하다.
- 단점
  - 매번 TCP/IP 연결을 새로 맺어야 한다.
  - 웹 브라우저로 사이트를 요청하면 HTML 뿐만 아니라 JS, css, 이미지등 다른 자원이 함께 다운로드 되어야 하는데, 각 과정마다 연결을 맺었다 끊으면 시간이 낭비된다.

  → HTTP 지속 연결로 이를 보완!

  HTTP 지속연결이란 한 번의 TCP 연결을 통해 여러 개의 HTTP 요청과 응답을 처리하는 방식으로 HTTP의 비 연결성과 어느 정도 타협하면서 성능을 개선하는 방법
- 수강신청, 티켓팅 같은 동시에 대용량 트래픽이 발생하는 업무같은 경우, 미리 사용자가 TCP연결을 진행 하도록 정적인 이벤트 페이지등을 만드는 방법을 쓰기도 한다.

HTTP 메시지

HTTP 메시지의 구조

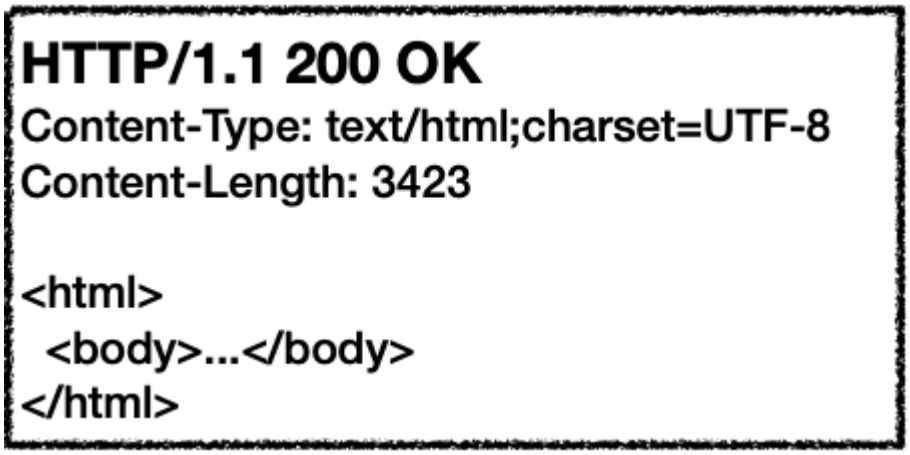


HTTP 요청 메시지

```
GET /search?q=hello&hl=ko HTTP/1.1
Host: www.google.com
```

- 시작라인
  - HTTP 메서드, 요청 대상, HTTP 버전
- 헤더
  - Host, User-Agent(클라이언트 정보), Accept, Authorization, Content-Type, Cookie등의 정보가 들어갈 수 있음

HTTP 응답 메시지



- start-line
  - HTTP 버전, 상태코드,상태 메시지(OK와 같이 사람이 이해할 수 있는 짧은 상태코드)
- HTTP 헤더
  - HTTP 전송에 필요한 모든 부가정보
  - 예) 메시지 바디의 내용, 메시지 바디의 크기, 압축, 인증, 요청 클라이언트(브라우저) 정보 등
- HTTP 메시지 바디
  - 실제 전송할 응답 데이터
  - HTML 문서, 이미지, 영상, JSON 등등 byte로 표현할 수 있는 모든 데이터 전송 가능

HTTP 메서드

- GET: 리소스 조회
  - 서버에 전달하고 싶은 데이터는 query(쿼리 파라미터, 쿼리 스트링)를 통해서 전달
  - 서버에 GET 메서드를 사용한 HTTP요청 메시지를 전달하면 서버는 응답메시지를 클라이언트에게 보내줌
  - 메시지 전달 → 서버 응답 → 응답 데이터 전달

- POST: 요청 데이터 처리, 주로 등록에 사용
  - **메시지 바디를 통해 서버로 요청 데이터 전달** 서버는 요청 데이터를 **처리**

- 메시지 바디를 통해 들어온 데이터를 처리하는 모든 기능을 수행한다.
- 주요 기능

  1. 새 리소스 생성(등록)

  서버가 아직 식별하지 않은 새 리소스 생성

  2. 요청 데이터 처리

  단순히 데이터를 생성하거나, 변경하는 것을 넘어서 프로세스를 처리해야 하는 경우
  예) 주문에서 결제완료 -> 배달시작 -> 배달완료 처럼 단순히 값 변경을 넘어 프로세스의 상태가 변경되는 경우 POST의 결과로 새로운 리소스가 생성되지 않을 수도 있음
  예) POST /orders/{orderId}/start-delivery (**컨트롤 URI**)

  3. 다른 메서드로 처리하기 애매한 경우

  예) JSON으로 조회 데이터를 넘겨야 하는데, GET 메서드를 사용하기 어려운 경우 애매하면 POST

- PUT: 리소스를 대체(덮어쓰기), 해당 리소스가 없으면 생성
  - 리소스가 있으면 리소스를 완전히 덮어쓴다.(주의해야함)
  - 클라이언트가 리소스 위치를 알고 URI를 지정하여 전송한다. (POST와 차이)
- PATCH: 리소스 부분 변경
  - 리소스 내의 데이터를 부분적으로 변경할 수 있음
- DELETE: 리소스 삭제
  - 리소스 내의 데이터를 부분적으로 삭제 가능
- 기타: HEAD, OPTIONS, CONNECT, TRACE

HTTP 메서드의 속성

- 안전
  - 호출해도 리소스를 변경하지 않는다.(단순 읽기)
  - GET,HEAD,OPTIONS,TRACE
- 멱등
  - 여러번 호출해도 같은 결과가 나오는가?

    - GET,HEAD,PUT,DELETE
    - ex) 자동 복구 매커니즘 : 서버가 정상 응답을 못주었을 때, 클라이언트가 같은 요청을 다시해도 되는가? → 멱등이면 다시 해도 됨
- 캐시가능
    - 응답 결과 리소스를 캐시해서 사용해도 되는가?
    - GET,HEAD,POST
    - 실제로는 GET,HEAD정도만 사용, POST는 본문까지 캐시 키로 고려해야 하는데 구현이 어려움

HTTP 상태코드

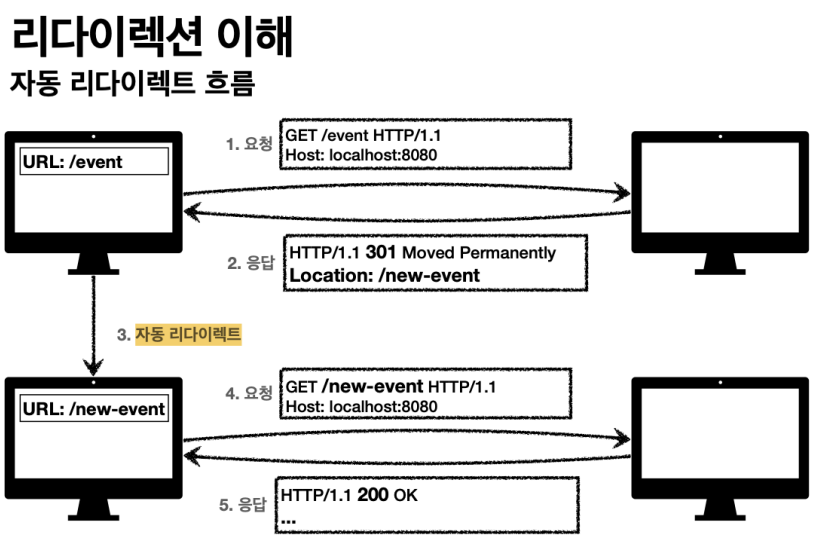
클라이언트가 보낸 요청의 처리 상태를 응답에서 알려주는 기능

- 상태코드의 종류
    - 1xx (Informational): 요청이 수신되어 처리중
    - 2xx (Successful): 요청 정상 처리
    - 3xx (Redirection): 요청을 완료하려면 추가 행동이 필요
    - 4xx (Client Error): 클라이언트 오류, 잘못된 문법등으로 서버가 요청을 수행할 수 없음
    - 5xx (Server Error): 서버 오류, 서버가 정상 요청을 처리하지 못함
- 모르는 상태 코드 처리
    - 클라이언트가 인식할 수 없는 상태코드를 서버가 반환하면 클라이언트는 상위 코드로 해석해서 처리한다. 미래에 새로운 상태 코드가 추가 되어도 클라이언트를 변경하지 않아도 된다.
    - ex) 299 → 2xx (Successful), 451 → 4xx (Client error)
- 2XX(Successful)
- 200 OK
- 201 Created: 요청을 성공해서 새로운 리소스가 생성됨(생성된 리소스는 응답의 Location 헤더 필드로 식별 ex) Location: /members/100)
- 202 Accepted : 요청이 접수되었으나 처리가 완료되지 않았음
- 204 No Content: 서버가 요청을 성공적으로 수행했지만, 응답 페이로드 본문에 보낼 데이터가 없음 ex) 웹 문서 편집기에서 save 버튼

3XX(Redirection)

- 리다이렉션의 개념

웹 브라우저는 3xx 응답의 결과에 Location 헤더가 있으면, Location 위치로 자동 이동 (리다이렉트)



-영구 리다이렉션 - 특정 리소스의 URI가 영구적으로 이동

ex) /members → /users

-일시 리다이렉션 - 일시적인 변경

ex) 주문 완료 후 주문 내역 화면으로 이동

- 영구 리다이렉션 (301,308)
  - 영구 리다이렉션
  - 301 - **Moved permanently**: 리다이렉트시 요청 메서드가 GET으로 변하고, 본문이 제거될 수 있음(MAY)
  - 308 - **Permanent Redirect:** 301과 기능은 같고 차이점은 리다이렉트시 요청 메서드와 본문 유지(처음에 POST를 보내면 리다이렉트도 POST유지)
  - 보통은 내용도 함께 변하는 경우가 많아서 301을 더 많이 사용. 둘다 많이 쓰이지는 않고 일시 리다이렉션을 더 많이 사용
- 일시적인 리다이렉션(302,303,307)
  - **리소스의 URI가 일시적으로 변경**
  - **따라서 검색 엔진 등에서 URL을 변경하면 안됨**
  - 302: **리다이렉트시 요청 메서드가 GET으로 변하고, 본문이 제거될 수 있음(MAY)**

    - MAY인 이유: 초기 SPEC에서 제대로 설명을 안해놔서 어떤 브라우저는 제거되게(대부분) 어떤 브라우저는 제거되지 않게 구현을 해놓음, 이후 SPEC이 바뀌어서 애매하게 설명.
      307,303을 만들어 기능을 확실히 해주면 된다.
  - 307: **리다이렉트시 요청 메서드와 본문 유지(요청 메서드를 변경하면 안된다. MUST NOT)**
  - 303 See Other**리다이렉트시 요청 메서드가 GET으로 변경**

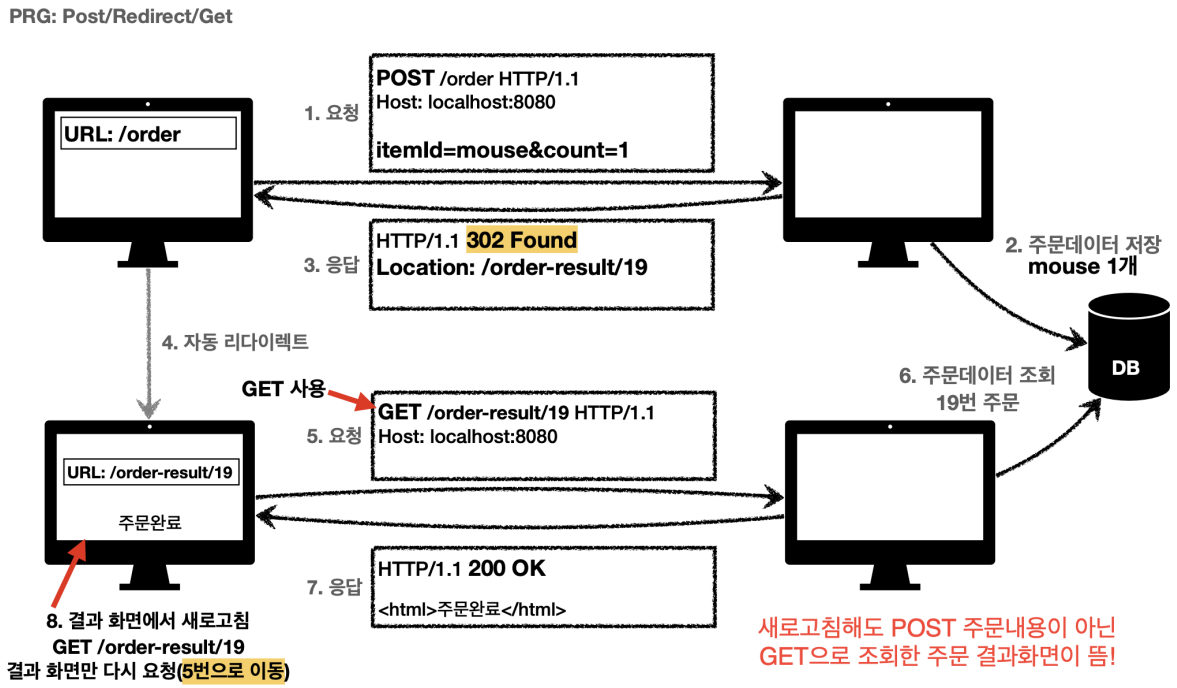
PRG(Post/Redirect/Get)

-POST로 주문 후 웹 브라우저를 새로고침하면 다시 요청되어 중복 주문이 될 수 있다. 이를 방지

-POST로 주문후에 새로 고침으로 인한 중복 주문 방지

-POST로 주문후에 주문 결과 화면을 GET 메서드로 리다이렉트

-새로고침해도 결과 화면을 GET으로 조회 → 중복 주문 대신에 결과 화면만 GET으로 다시 요청



- **기타 리다이렉션(대부분 304)**

-캐시를 목적으로 사용
-클라이언트에게 리소스가 수정되지 않았음을 알려준다. 따라서 클라이언트는 로컬PC에 저

장된 캐시를 재사용한다. (캐시로 리다이렉트 한다.)
-304 응답은 응답에 메시지 바디를 포함하면 안된다. (로컬 캐시를 사용해야 하므로) 조건부 GET, HEAD 요청시 사용

4xx(Client Error)

-클라이언트의 요청에 잘못된 문법등으로 서버가 요청을 수행할 수 없음

-오류의 원인이 클라이언트에 있음

-클라이언트가 이미 잘못된 요청, 데이터를 보내고 있기 때문에, 똑같은 재시도가 실패함!!

→클라이언트는 요청 내용을 다시 검토하고, 보내야함

- **401 Unauthorized**
  - **클라이언트가 해당 리소스에 대한 인증이 필요함**
  - 인증(Authentication): 본인이 누구인지 확인, (로그인)
  - 인가(Authorization): 권한부여 (ADMIN 권한처럼 특정 리소스에 접근할 수 있는 권한, 인증이 있어야 인가가 있음)
  - 오류 메시지가 Unauthorized 이지만 기능은 인증쪽에 더 가까움(이름이 아쉬움)
- **403 Forbidden**
  - 서버가 요청을 이해했지만 승인을 거부함
  - 주로 인증 자격 증명은 있지만, 접근 권한이 불충분한 경우
- **404 Not Found**
  - 요청 리소스가 서버에 없음
  - 또는 클라이언트가 권한이 부족한 리소스에 접근할 때 해당 리소스를 숨기고 싶을 때

5xx(Server Error)

서버 문제로 오류 발생, 서버에 문제가 있기 때문에 재시도 하면 성공할 수도 있음(복구가 되거나 등등)

5XX는 정말 서버에 문제가 있을 때 내야한다. 비즈니스 로직상 문제가 있는 경우에는 5XX 에러를 내면 안된다!

- **500 Internal Server Error**
  - 서버 문제로 오류 발생, 애매하면 500 오류
- **503 Service Unavailable**

- 서버가 일시적인 과부하 또는 예정된 작업으로 잠시 요청을 처리할 수 없음
- Retry-After 헤더 필드로 얼마뒤에 복구되는지 보낼 수도 있음

HTTPS

- HTTPS의 등장배경

우선 HTTP가 뭔지 되짚어 보자. HTTP는 HyperText Transfer Protocol의 약자로, 웹 브라우징에서 데이터를 주고받기 위한 기본적인 프로토콜이다. HTTPS는 여기에 Secure를 뜻하는 S가 붙었으며 이는 HTTP의 보안 강화 버전이라 할 수 있다. 그렇다면 HTTP가 가지고 있는 보안 문제는 무엇일까? HTTP는 데이터를 평문으로 전송하기 때문에 중간에서 가로채거나 변조될 수 있는 위험이 있다. 이로 인해 사용자의 민감한 정보가 노출될 수 있는 위험이 있다. 이와 같은 문제를 해결하기 위해 HTTPS가 등장하였으며, 현재 대부분의 웹사이트는 HTTPS를 이용하고 있다.

- HTTP vs HTTPS

HTTP와 HTTPS의 가장 큰 차이점은 HTTPS는 데이터를 평문으로 전송하고, HTTPS는 데이터를 암호화하여 전송한다는 것이다. 따라서 공격자가 중간에 데이터를 가로채도 이를 읽을 수 없다.

특징	HTTP	HTTPS
프로토콜	HyperText Transfer Protocol	HyperText Transfer Protocol Secure
보안	데이터 암호화 없음	데이터 암호화 (SSL/TLS 사용)
포트 번호	80	443
보안 인증서	필요 없음	SSL/TLS 인증서 요구
주소 표시	http://	https://
사용 사례	보안이 중요하지 않은 데이터 전송	개인정보, 결제 등 민감한 데이터 전송