**Rest API 학습**

최동준

## **HTTP 통신**

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

* 텍스트 기반의 통신 규약으로 인터넷에서 데이터를 주고 받을 수 있는 프로토콜
* 클라이언트 서버 모델을 따르며 TCP/IP 위에 작동
  + 클라이언트 서버 모델
    - 클라이언트가 HTTP 메시지를 통해 서버에 요청(request)을 보냄
    - 서버가 요청에 대한 결과를 만들어서 응답(response) 전송
    - 응답을 받은 클라이언트가 응답 결과를 열어서 동작하게됨.
    - 클라이언트가 서버에 요청을 보내기 전에 먼저 handshake로 연결을 맺어야함.
  + HTTP request 구조

|  |
| --- |
| GET /index.html HTTP/1.1  Host: example.com  User-Agent: Mozilla/5.0  Accept-Language: ko-KR |

* 첫 줄: HTTP 프로토콜 버전 정보와 HTTP 상태 코드
* 둘째 줄: key:value 쌍은 헤더
  + HTTP response 구조

|  |
| --- |
| HTTP/1.1 200 OK  Date: Sat, 09 Oct 2023 14:28:02 GMT  Server: Apache  Content-Type: text/html  <html>  ...  </html> |

* + - 응답의 헤더는 브라우저가 필요한 정보를 전달해줌
    - 응답의 바디는 브라우저가 요청한 데이터
    - 위 예시는 html을 리턴해주는 것
  + HTTP와 HTTPS의 차이점
    - HTTP에서는 데이터를 암호화하지 않고 전송하기 때문에 제 3자가 데이터를 가로채고 읽을 수 있음.
    - 보안을 강화하기 위해서는 HTTPS를 사용해야함
    - HTTPS에서는 브라우저와 서버가 데이터를 전송하기 전에 안전하고 암호화된 연결을 생성함.
  + 상태 코드
    - 1xx(정보): 요청을 받았으며 프로세스를 계속 진행
    - 2xx(성공): 요청을 성공적으로 받았으며 인식했고 수용했다.
    - 3xx(리다이렉션): 요청 완료를 위해 추가적인 동작 필요. 주로 서버의 주소 또는 요청한 URI의 웹 문서가 이동되었으니 그 주소로 다시 시도하라
    - 4xx(클라이언트 오류): 요청의 문법이 잘못되었거나 요청을 처리할 수 없다.
    - 5xx(서버 오류): 서버에서의 오류가 발생했다.
  + HTTP 메서드
    - GET: 리소스를 조회
    - POST: 데이터 추가, 등록
    - PUT: 리소스 대체, 수정 / 해당 리소스가 없으면 새롭게 생성
    - DELETE: 리소스 삭제
    - PATCH: 리소스 부분 변경(수정)
    - HEAD: GET과 동일하나. HTTP 메시지의 body부분을 제외하고 조회
    - OPTIONS: 서버와 클라이언트가 통신하기 위한 통신 옵션을 확인하기 위함
      * 서버가 어떤 메서드, 헤더, content-type을 제공하는지 알 수 있음.
    - CONNECT: 대상 자원으로 식별되는 서버에 대한 연결 요청
    - 주로 사용하는건 GET, POST, PUT, DELETE, PATCH

### **REST API**

* 구성 요소
  + 자원(Resource): HTTP URI
  + 자원에 대한 행위(Verb): HTTP method
  + 자원에 대한 행위의 내용(Representations): HTTP Message Pay Load
* 제한 조건 6가지
  + 인터페이스 일관성: 일관적인 인터페이스로 분리되어야 한다.
  + 무상태: 서버는 클라이언트 요청과 관련된 데이터를 저장할 수 없다. 그래서 각 요청에는 처리에 필요한 모든 정보가 포함되어야 한다.
  + 클라이언트-서버 분리: 클라이언트와 서버가 서로 완전히 독립적이어야 한다. 클라이언트가 알아야 하는 유일한 정보는 요청 리소스의 URI이다. 서버와 다른 방식으로 통신할 수 없다. 서버도 요청된 데이터를 전달하는 것 외에는 클라이언트를 수정해서는 안된다.
  + 캐시 가능성: 가능한 경우 클라이언트나 서버 측에서 리소스를 캐시할 수 있어야 한다.
  + 계층화된 시스템 아키텍쳐: 호출과 응답이 서로 다른 계층을 거친다. 클라이언트는 보통 서버에 직접 연결되었는지, 중간 서버를 거치는지 알 수 없다. 중간 서버는 로드 밸런싱이나 공유 캐시 기능을 제공할 수 있다.
  + 코드 온디맨드(선택사항): 일반적으로 정적 리소스를 전송하지만, 경우에 따라 응답에 실행 코드가 포함될 수도 있다.
* 작동 방식
  + HTTP 요청을 통해 리소스 내에서 레코드를 생성하고 CRUD같은 표준 데이터 베이스 기능을 수행한다.
  + 예시
    - GET 요청을 사용하여 레코드를 검색
    - POST 요청은 새 레코드를 생성
    - PUT 요청은 레코드를 업데이트
    - DELETE 요청은 레코드를 삭제
  + 잘 설계된 REST API는 HTTP 기능이 내장된 웹 브라우저에서 실행되는 웹사이트와 유사하다.
  + 특정 순간 또는 타임스탬프의 리소스 상태를 리소스 표현이라고 하는데, 이 정보는 JSON, HTML, XLT, Python, PHP 또는 일반 텍스트를 포함한 거의 모든 형식으로 클라이언트에 전달될 수 있다. JSON은 사람과 기계 모두 읽을 수 있고, 프로그래밍 언어에 구애받지 않아 널리 사용된다.
  + 요청 헤더 및 매개 변수는 메타 데이터, 권한 부여, URI, 캐싱, 쿠키 등과 같은 중요한 식별자 정보를 포함하기 때문에 REST API 호출에서도 중요하다. 요청 헤더 및 응답 헤더는 기존 HTTP 상태 코드와 함께 잘 설계된 REST API 내에서 사용된다.
* 설계 규칙
  + URI는 동사보다는 명사를, 대문자보다는 소문자를 사용해야한다.
  + 마지막에 슬래시를 포함하지 않는다.
  + 언더바 대신 하이픈을 사용한다.
  + 파일 확장자는 URI에 포함하지 않는다.
  + 행위를 포함하지 않는다.
* RESTful
  + REST의 원리를 따르는 시스템을 의미한다. REST를 사용했다고 해서 모두가 RESTful한 것은 아니다. REST API의 설계 규칙을 올바르게 지킨 시스템을 RESTful하다고 할 수 있다.
* 장점
  + HTTP 프로토콜의 인프라를 그대로 사용해서 REST API를 위한 별도의 인프라를 구축할 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
  + HTTP method 형태가 제한적이다.
  + 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
  + 구형 브라우저에서 호환이 되지 않아 지원해주지 못하는 동작이 많다.