REST API ⌨️

`모든 웹은 API` 포스트를 통해 API가 어떤 역할을 하는지 배울 수 있었고 추가로 최신 웹은 REST API로 구성되어 동작하고 있다는걸 간단하게 알아보았는데,
API면 API지 Representational State Transfer API 라는 건 도대체 뭘까?

여러 블로그를 찾아보고 생각을 종합해 보니 REST API는
`클라이언트와 서버 간에 데이터를 주고, 받을 때 사용하는 규칙`으로 큰 틀에서 봤을 때 API가 갖는 특성과 일치한다.

이왕 웹 사이트를 만들고 데이터를 주고받을 거 정해진 규칙 대로 잘 만들어 잘 쓰자는 게 목표인데
여기에 REST를 더 해 어떻게 잘 만들고 어떻게 잘 쓸지를 상세하게 정의한 API TODO 리스트라고 생각 할 수 있겠다.

RESTful하지 못하다? 😾

예를 들어보자 나는 카페를 운영하고 있고 고객이 카페라테에 우유를 빼달라고 요청했다.
메뉴판에 카페라테는 에스프레소+우유+(얼음)이 들어간다고 명시해놨지만, 고객의 요청을 무시할 수 없어 우유 빠진 카페라테를 제공했다면
고객(REST API)은 메뉴판(REST 권장 사항)의 권장 사항을 준수하지 않았고, 나(Server)는 요청에 맞지만 표준엔 틀린 음료(데이터)를 제공하게 된다.

다시 말해 "RESTful 하지 않다."라는 말은
REST API가 웹 기술의 표준으로 정의된 REST의 권장 사항을 준수하지 않았다는걸 뜻하고 이는 일관성이 없고 유지 보수성이 떨어지는 구조를 가지게 된다.

REST 표준 구조 ⌨️

Client - Server
REST 아키텍처에서 client-server 모델은 클라이언트와 서버가 서로 분리된 역할을 하는것을 의미한다.
클라이언트는 UI를 통해 요청을 보내고, 서버는 요청에 대한 응답을 제공하는 것으로 각자 독립적이며 역할과 책임이 나뉜다.

stateless
HTTP는 기본적으로 무 상태를 지향한다.
이는 클라이언트 서버 간의 각 요청에 대해 상태 정보를 저장하지 않는 것을 의미하고 그러므로 REST API는 모든 정보가 요청/응답 메시지 안에 포함되어야 하며,
그 결과 서버는 요청에 대한 정보를 저장하거나 기억할 필요가 없어지게 된다.

이를 통해 서버의 역할은 줄어들게 되고 확장성을 높이는 데 도움을 준다.
무 상태의 이점으로 견고성과 확장성은 좋아졌지만, 클라이언트는 매 요청 서버에게
전체 정보를 전달해야 하는 책임이 생기게 되었고 바로 이 부분이 많은 REST API를 RESTful 스럽지 못하게 만드는 주요 포인트이다.

cache
먼저 cache에 대해 간단히 짚고 넘어가 보자

"캐시를 로컬에 저장한다"라는 말은 클라이언트가 서버로부터 받은 응답 메시지(HTML, JSON, XML ..)를 클라이언트의 로컬 저장소(운영체제의 파일 시스템 등)에 저장하는 것을 의미한다.

이렇게 저장된 정보는 클라이언트가 같은 요청을 수행할 때,
서버를 통하지 않고 로컬의 저장 정보를 사용하고 이를 통해 서버의 부하를 줄이고 클라이언트의 응답 속도는 향상되는 이점을 가질 수 있다.

cache 기술은 HTTP 프로토콜의 header 정보를 이용해 유효 기간을 설정하거나,
캐시 사용 여부를 결정하는 등 서버와 클라이언트 간의 규칙을 결정할 수 있게 된다.

REST API에서도 cache는 같은 역할을 한다.

Uniform Interface

Uniform Interface는 REST 아키텍처의 핵심 요소로 REST API의 각 자원에 대해 공통으로 사용할 수 있는 인터페이스를 제공한다.

Resource Identifier(URI)

- 모든 소스는 고유한 URI를 제공하여 접근할 수 있도록 한다.

Representations

- HTML, JSON, XML 등의 데이터 형식으로 자원 상태를 표현할 수 있고 이를 통해 생성, 조회, 수정, 삭제가 가능해야 한다.

Self-descriptive messages

- 클라이언트가 소스에 대해 요청을 할 때, 소스의 표현, 상태, 헤더 등에 대한 정보를 포함하는 메시지를 제공하고 이 메시지를 통해 스스로를 설명 할 수 있어야 한다.
    - **서버나 클라이언트가 변경되더라도 오가는 메시지는 언제나 스스로를 설명 할 수 있으므로 확장할 수 있는 커뮤니케이션이 가능해진다.**

HATEOS(Hypermedia as the Engine of Application State)

- HATEOS는 애플리케이션의 상태를 유지하는 기능을 제공한다. 소스에 대한 요청에 대해 응답과 관련된 다른 소스의 URI를 포함하여 이동할 수 있어야 한다.
    - **애플리케이션의 상태는 Hyperlink를 통해 전이되어야 한다.**
        - 어디서 어디로 전이할 수 있는지 미리 결정되지 않는다.

> 어떤 상태로 전이가 완료되고 나서야 그다음 전이될 수 있는 상태가 결정된다. 이는 링크가 동적으로 변경될 수 있음을 뜻한다.

사실 구두로 표현된 말이 와닿지 않고 어떻게 적용할 것인지 생각하면 막막한 느낌이 든다.
이런 복잡한 규칙을 가진 Uniform Interface를 사용해야 하는 이유가 뭘까?

"How do I improve HTTP without breaking the Web?"

REST 개념을 처음 세상에 발표한 건 Roy T. Fielding이라는 사람으로
웹 페이지와 HTTP의 유기성에 대한 고민을 시작으로 발전하게 되었다. 이에 대한 결론이
앞서 살펴본 Uniform Interface로 규칙을 준수했을 때 우리가 얻을 수 있는 이점은 크게 다음과 같다.

- 서버와 클라이언트는 각각 독립적으로 발전할 수 있다.
- 서버의 기능이 변경되어도 클라이언트는 변화가 없다.
- 웹을 손상 시키지 않고 HTTP를 개선할 수 있다.

Uniform Interface와 이점의 연관 관계가 와닿지 않을 수 있는데, 우리는 항상 그 결과물을 보고 있다.

REST를 가장 REST스럽게 사용한 웹 브라우저가 대표적인 예시로 크롬을 업데이트하지 않아도 나는 당장 크롬에 접속해 사용할 수 있고
구버전을 사용하더라도 UI가 깨지는 등의 불편함은 생길 수 있지만 기능 사용에 문제는 없다.

어떻게 그럴 수 있을까?

Uniform Interface를 지킨 REST API는 클라이언트와 서버 사이에 공통된 인터페이스를 제공하고, 통신의 규칙을 정의했기 때문이다.

그 결과 클라이언트 서버 사이에 데이터 형식, 메시지 구조, 상태 전이 등이 일관적으로 관리되고, 개발자는 좀 더 쉽게 API를 개발하고 유지 보수 할 수 있는 환경을 받을 수 있게 되었다.

Layered System⌨️

단어 그대로의 의미를 가진다. REST API는 클라이언트와 서버의 관계를 다양한 계층으로 구분해 관리하는 구조를 가질 수 있다.

1. **Client Layer**
   - 클라이언트 애플리케이션이 포함되는 계층
2. **Cache Layer**
   - REST API의 응답을 캐싱하여, 빠른 응답 속도를 제공하는 계층
3. **Application Layer**
   - REST API의 비즈니스 로직을 구현하는 계층
4. **Transport Layer**
   - REST API의 통신 규칙을 관리하는 계층

이 계층 구조를 통해 클라이언트와 서버의 의존성은 줄어들고 각 계층의 역할과 책임이 명확해져 유지 보수와 확장이 쉬워진다.

Code-On-Demand (Optional) ⌨️

서버에서 코드를 클라이언트로 보내고 클라이언트가 이를 다운 받아 실행할 수 있음을 의미한다. 쉽게 말해 자바스크립트 코드를 클라이언트가 다운로드 할 수 있다는 말인데 이를 통해 우리는 웹 사이트를 보다 동적으로 접근할 수 있다.

단점과 보안 방법

1. REST API는 기본적으로 HTTP 프로토콜을 기반으로 일부 애플리케이션에서 보안 문제가 발생 할 수 있다.
   예를 들어 HTTP를 사용하면 데이터가 평문으로 전송되므로 중간에서 데이터를 가로챌 수 있다.

   HTTPS의 SSL/TLS를 사용해 데이터를 암호화하고 인증 한다.

2. 서버에서 처리할 수 있는 모든 데이터가 클라이언트로 전송되기 때문에 대용량 데이터를 다루는 앱의 경우 처리 속도가 느려진다.

   OAuth나 JWT를 사용해 인증과 권한 부여를 통해 유효한 사용자만 데이터에 접근할 수 있게 한다. (접근 제한)

3. 데이터를 클라이언트 측에서 직접 처리할 수 있으므로 보안상 취약한 클라이언트 앱에서는 데이터 변조나 해킹의 문제가 발생할 수 있다.

   API 엔드 포인트를 보호하기 위해 웹 방화벽, 인트루전 디텍션 시스템(IDS), 인트루전 프리벤션 시스템(IPS)를 사용해
   DDos, 인젝션, 스크립트 공격 등을 방어하는 보안 솔루션을 구현한다.

정리 ⌨️

- REST API는 API에 상세한 명세를 더 해 구조화 한 것이다.
- REST API는 REST 표준을 따르는데, 이를 준수하지 못 했다면 RESTful 하지 못한 API가 된다.
- 계층 구조를 가져 독립성과 확장성 유지 보수가 뛰어나다.
- 알려진것과 다르게 REST의 표준 구조는 까다롭다.