RESTful API는 웹 서비스 개발에서 자주 사용되는 아키텍처 스타일로 클라이언트와 서버 간의 통신을 간단하고 효율적으로 만드는 방식이다. REST(Representational State Transfer)는 웹의 기본 원칙에 기반하며 HTTP 프로토콜을 활용해 클라이언트가 서버의 자원에 접근하도록 설계되었다.

REST의 기본 원칙

1. 클라이언트-서버 아키텍처 : 클라이언트와 서버는 서로 독립적으로 동작하며, 클라이언트는 사용자 인터페이스를 관리하고 서버는 데이터 저장과 처리 로직을 관리한다.
2. 무상태성 : 서버는 각 요청을 독립적으로 처리하며 이전 요청에 대한 정보를 저장하지 않는다. 모든 요청에는 필요한 모든 정보가 포함되어야 한다.
3. 캐시 처리 가능 : 응답을 캐싱할 수 있도록 메시지에 캐싱 가능 여부를 명시한다. 캐싱을 통해 네트워크 트래픽을 줄이고 응답 속도를 향상시킬 수 있다.
4. 계층화된 시스템 : 클라이언트는 서버와 직접 통신할 수도 있고 중간 계층(로드 밸런서, 프록시)을 거쳐 통신할 수도 있다.
5. 인터페이스의 일관성 : RESTful 시스템에서는 모든 리소스에 대해 일관된 방식으로 접근할 수 있어야 한다. 이 원칙은 URL 구조, HTTP 메서드, 상태 코드 등이 표준화되어야 함을 의미한다.
6. 요청의 표현 : 클라이언트는 리소스 자체가 아닌 그 표현을 요청한다. 서버의 데이터베이스에 저장된 리소스는 JSON, XML 등 다양한 형식으로 표현될 수 있다.

RESTful API의 구성 요소

RESTful API는 주로 HTTP 프로토콜과 함께 사용된다.

리소스(Resource) : 모든 것(데이터, 객체 등)을 리소스로 간주하며 각 리소스는 고유한 URL(엔드포인트 : 웹 애플리케이션에서 클라이언트가 서버에 요청을 보내는 특정 URL 경로)로 식별된다.
예시 : `https://api.sample.com/users/123` (사용자 ID가 123인 사용자에 대한 리소스)

1. HTTP 메서드 : 리소스에 대해 수행할 작업을 정의하는 데 사용된다.
    1. GET : 리소스를 조회
    2. POST : 새로운 리소스를 생성
    3. PUT : 리소스를 업데이트
    4. DELETE : 리소스를 삭제
    5. PATCH : 리소스의 일부를 업데이트
    6. 그 외 메서드 : HEAD (GET 메서드와 유사하지만 응답 본문 대신 헤더 정보만 받는다), OPTIONS (서버가 지원하는 HTTP 메서드를 확인), TRACE (요청이 서버를 거치면서 어떤 변화를 겪는지 추적, 디버깅 목적), CONNECT (터널링을 설정하여 다른 프로토콜을 통해 통신)

2. HTTP 상태 코드 : 서버가 클라이언트의 요청에 대해 어떤 결과를 반환하는지 나타낸다.
    1. 200 OK : 요청이 성공
    2. 201 Created : 새로운 리소스가 성공적으로 생성
    3. 400 Bad Request : 잘못된 요청
    4. 401 Unauthorized : 인증이 필요
    5. 404 Not Found : 요청한 리소스를 찾을 수 없음
    6. 500 Internal Server Error : 서버에서 문제가 발생
3. URL(Uniform Resource Locator) : RESTful API에서는 각 리소스가 고유한 URL로 식별됩니다. URL은 명사로 표현되며 리소스에 대한 위치를 나타낸다.
    1. 예: `https://api.sample.com/users/` (모든 유저 목록을 가져옴)
4. 헤더(Headers) : 클라이언트와 서버 간의 추가 정보를 전송할 때 사용된다. 콘텐츠 타입(Content-Type)을 지정하여 요청이나 응답 데이터의 형식을 나타낼 수 있다.
    1. Content-Type : `application/json` , `application/xml` 등
5. 페이로드(Payload) : REST API에서 POST, PUT, PATCH와 같은 메서드를 사용할 때 클라이언트(보통 웹 브라우저나 앱)에서 서버로 데이터를 보내야 하는 경우가 많다. 이때 보내는 데이터 자체를 페이로드라고 한다.

RESTful API의 설계 원칙

RESTful API는 일관되고 사용하기 쉬운 설계가 중요하다.

1. 명확한 URI(Uniform Resource Identifier): 각 리소스는 명확한 URI로 식별된다. URI는 간결하고 이해하기 쉽게 설계해야 한다.
   예시 : `/customers/456/invoices` (고객 ID가 456인 사용자의 청구서 목록)
2. 일관된 HTTP 메서드 사용 : GET, POST, PUT, DELETE 등의 HTTP 메서드를 일관되게 사용하여 API의 사용성을 높인다.
   예시: `GET /orders/789` (주문 ID가 789인 주문의 세부 정보 조회)
3. 응답 데이터의 구조 : API의 응답 데이터는 클라이언트가 쉽게 이해하고 처리할 수 있는 구조여야 한다. 일반적으로 JSON 형식을 사용한다.
   예시 : `{"id": 789, "status": "shipped", "items": [{"product_id": 321, "quantity": 2}]}`
4. 필터링, 정렬, 페이징 : 대규모 데이터셋에서 유용하게 사용된다. 특정 조건에 맞는 데이터만 가져오거나 데이터를 페이지별로 나누어 가져올 수 있도록 설계한다.
   예시 : `/employees?department=sales&sort=name&limit=20&page=3` (영업 부서 직원들을 이름순으로 정렬한 세 번째 페이지에서 20명 가져오기)
5. 버전 관리 : API의 버전 관리는 중요한 기능이다. 새로운 기능 추가나 변경 사항이 기존 클라이언트에 영향을 미치지 않도록 URL에 버전을 포함하는 방식이 일반적이다.
   예시 : `/api/v2/products/654` (API 버전 2에서 제품 ID가 654인 제품의 정보 조회)

마무리

RESTful API는 현대 웹 개발에서 필수적인 요소이다. 올바른 설계와 구현은 애플리케이션의 확

장성(사용량에 대해 어떻게 대응할 수 있는지를 나타내는 속성)과 유지보수성(지하고 관리하는 데 얼마나 용이한지를 나타내는 속성)을 크게 향상시킬 수 있다.